TrajectoryTest: A Trajectory-Specific Test Input Prioritization Technique and Empirical Evaluation

XUEQI DANG, University of Luxembourg, Luxembourg
YINGHUA LI*, Nanjing University of Science and Technology, China
WENDKÛUNI C. OUÉDRAOGO, University of Luxembourg, Luxembourg
MIKE PAPADAKIS, University of Luxembourg, Luxembourg
JACQUES KLEIN, University of Luxembourg, Luxembourg
TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg
YVES LE TRAON, University of Luxembourg, Luxembourg

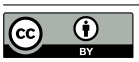
Trajectory prediction models have become increasingly vital in various safety-critical domains, such as healthcare, maritime surveillance, and intelligent transportation systems. These applications utilize machine learning models to predict the operational state of a moving agent based on its motion trajectory. Despite their widespread deployment in these domains, testing such models remains a significant challenge due to the high cost of manual labeling, further exacerbated by the large scale and domain-specific characteristics of trajectory datasets. Test input prioritization has emerged as a promising solution to address the labeling cost issue, aiming to identify potentially misclassified inputs early to accelerate debugging and improve testing efficiency. The current state-of-the-art test prioritization technique that can be applied to trajectory prediction tasks is MLPrior. However, MLPrior has the following limitations when applied to trajectory prediction scenarios: 1) MLPrior relies on rich input features, which are limited in trajectory data. Trajectory inputs typically contain only basic spatiotemporal coordinates (e.g., latitude, longitude, timestamps), offering insufficient attribute richness. 2) MLPrior inherits the limitations of learning-based approaches. When the training data are imbalanced, their effectiveness can be significantly reduced. To overcome these limitations, we propose TrajectoryTest, a trajectory-specific test prioritization strategy that integrates trajectory-related information with MLPrior-derived representations to rank test inputs. Moreover, TrajectoryTest employs an adaptive strategy that switches between learning-based and uncertainty-based ranking depending on the model's prediction error ratio. We conduct a comprehensive empirical study that confirms the limitations of MLPrior in trajectory prediction scenarios and demonstrates that TrajectoryTest outperforms all existing test input prioritization techniques, including the state-of-the-art approach MLPrior, multiple confidence-based methods, and the baseline random selection. The experimental results show that TrajectoryTest achieves improvements ranging from 7.03% to 9.56% over MLPrior (the state-of-the-art method) and confidence-based approaches on natural datasets, and from 6.71% to 9.94% on noisy datasets.

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

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

*Corresponding author.

Authors' addresses: Xueqi Dang, xueqi.dang@uni.lu, University of Luxembourg, Luxembourg; Yinghua Li, yinghua.li@njust.edu.cn, Nanjing University of Science and Technology, China; Wendkûuni C. Ouédraogo, wendkuuni.ouedraogo@uni.lu, University of Luxembourg, Luxembourg; Mike Papadakis, michail.papadakis@uni.lu, University of Luxembourg, Luxembourg; Jacques Klein, jacques.klein@uni.lu, University of Luxembourg, Luxembourg; Tegawendé F. Bissyandé, tegawende.bissyande@uni.lu, University of Luxembourg, Luxembourg; Yves Le Traon, yves.letraon@uni.lu, University of Luxembourg, Luxembourg.



1 INTRODUCTION

Trajectory prediction has become increasingly critical in a wide range of application domains, including maritime monitoring [5, 51, 91], human activity recognition [23, 68], and intelligent transportation systems [54, 101]. These applications leverage machine learning models to predict the operational state of a moving agent based on its motion trajectory. A trajectory is typically represented as a time-ordered sequence of location points. Trajectory prediction plays a vital role in many safety-critical applications and has attracted significant attention from both academia and industry. For example, in the healthcare domain, trajectory information can be used to monitor whether an elderly person's current state or location is safe [80]. In the maritime domain, predicting the trajectory of certain fishing vessels can help identify suspicious or illegal activities such as unauthorized fishing in restricted zones [67].

While trajectory prediction models are increasingly deployed in safety-critical systems, ensuring their correctness and reliability remains a major challenge. Testing is widely recognized as one of the most fundamental approaches to ensuring the quality of machine learning models [93]. However, a significant challenge in testing trajectory prediction models is the high labeling cost issue: verifying whether the predictions for the trajectories are correct can be both expensive and time-consuming.

This challenge arises from several factors: 1) manual annotation remains the primary approach for labeling; 2) trajectory datasets can be large-scale, increasing labeling effort; 3) domain-specific expertise can be required to interpret temporal motion patterns. For instance, determining whether a predicted maritime trajectory accurately captures the future movements of a vessel typically requires expert analysis of vessel behavior and manual inspection of its motion over time. These factors make exhaustive labeling of large test datasets impractical.

Recent research has focused on ***test input prioritization*** to address the labeling cost problem in deep learning systems [24, 83]. These techniques aim to rank test inputs such that those most likely to reveal incorrect model behavior are prioritized for labeling. By identifying misclassified inputs earlier, developers can focus their efforts on the most critical test cases and accelerate the debugging process, thereby improving the overall efficiency of testing.

Currently, mainstream test prioritization methods can be categorized into coverage-based [57, 63], confidence-based [24, 84], and learning-based approaches [15, 47, 83]. Coverage-based methods adapt traditional software testing techniques to DNN testing and leverage coverage of neurons for test prioritization. Confidence-based methods prioritize test inputs where the model is more uncertain in its predictions. For example, DeepGini [24] leverages the Gini score to quantify the model's prediction uncertainty. Learning-based methods construct a feature vector for each test input and feed it into a learning-to-rank model to estimate the likelihood of misclassification, thereby enabling test input prioritization.

At present, the state-of-the-art test prioritization method adaptable to trajectory prediction scenarios is MLPrior [15], a learning-based approach specifically designed for classical machine learning models. Since MLPrior is designed for tabular data, it can be well-suited for trajectory prediction tasks, which are also tabular in nature. Another type of method that can be adapted to trajectory prediction scenarios is the confidence-based approach. Compared with learning-based methods, confidence-based methods offer a lightweight and effective solution for test input prioritization.

However, through an analysis of MLPrior's underlying rationale, we identify two key limitations that affect its performance in trajectory prediction tasks:

- **MLPrior relies on rich input features, which are limited in trajectory data.** MLPrior prioritizes test inputs by constructing a composite feature vector that integrates three types

of information. These vectors are used to train a learning-to-rank model that estimates the likelihood of misclassification. However, in the context of trajectory prediction, input features are typically limited to basic spatial-temporal coordinates (e.g., latitude, longitude, and timestamps), offering limited attribute richness. As a result, MLPrior's mechanisms become less effective and stable.

- **MLPrior inherits the limitations of learning-based approaches.** As a learning-based prioritization method, MLPrior relies on training a ranking model using labeled test outcomes. However, when the target model achieves high accuracy, the training set for the ranking model can become highly imbalanced. This is because: training labels for the ranking model are determined based on whether a test case is misclassified (labeled as 1) or correctly classified (labeled as 0). If only a small number of test inputs are misclassified, the proportion of positive samples will be extremely low, resulting in training set imbalance. Such imbalance can significantly reduce the learning effectiveness of the ranking model, thus decreasing the prioritization effectiveness of learning-based methods.

To address these limitations, we propose TrajectoryTest, a test prioritization strategy specifically designed for trajectory prediction models. The core idea is to 1) integrate trajectory-aware information with features derived from MLPrior to rank test cases and prioritize potentially misclassified tests, and 2) design an automated path selection approach based on the limitations of learning-based methods, overcoming the shortcomings of existing approaches.

To validate the limitations of MLPrior and demonstrate the effectiveness of our proposed approach, TrajectoryTest, we conducted a comprehensive empirical study across various trajectory prediction scenarios. The experimental results confirm that MLPrior shows relatively unstable and suboptimal performance on both natural and noisy trajectory prediction datasets. Moreover, TrajectoryTest is shown to outperform all existing test input prioritization techniques, including MLPrior. On natural datasets, TrajectoryTest achieves improvements ranging from 7.03% to 9.56% over MLPrior (the state-of-the-art method) and confidence-based approaches. On noisy datasets, the improvements range from 6.71% to 9.94%. Finally, we conduct an ablation study, confirming that each individual component (i.e., trajectory information features and MLPrior-derived features) contributes to the overall effectiveness of TrajectoryTest. To facilitate further research, we have made our dataset, results, and tools publicly available on Zenodo[1].

Our contributions are summarized as follows:

- **Empirical Study** We conduct the first empirical study to evaluate the effectiveness of existing test input prioritization methods, including the state-of-the-art method MLPrior and several confidence-based techniques, in the context of trajectory prediction. Our results reveal their limitations in handling test prioritization under both natural and noisy trajectory prediction scenarios.
- **Approach** We propose TrajectoryTest, a novel trajectory-specific test prioritization strategy that integrates trajectory-aware features with the information from MLPrior to improve test prioritization performance.
- **Evaluation** We compare TrajectoryTest with the state-of-the-art method MLPrior, as well as several confidence-based and baseline techniques. The results demonstrate that TrajectoryTest outperforms MLPrior and confidence-based approaches, achieving improvements ranging from 7.03% to 9.56% on natural datasets and from 6.71% to 9.94% on noisy datasets.
- **Ablation Study** We validate the contribution of each individual component within TrajectoryTest through an ablation study, confirming that each component contributes to its overall effectiveness.

[1] https://zenodo.org/records/17035802

2 BACKGROUND

In this section, we first introduce the basic concepts of trajectory prediction and its applications in intelligent systems, and then present the existing approaches for test input prioritization in DNNs.

2.1 Machine Learning for Trajectory Prediction

Trajectory prediction has emerged as a crucial research frontier in the area of intelligent systems [55, 68, 82]. It focuses on predicting the operational state of a moving agent based on its motion trajectory. For example, in the domain of fishing vessel trajectory analysis [87], the task involves predicting the vessel's operational status (e.g., "trawling") based on its historical trajectory, which is represented as tabular data consisting of attributes such as position (x, y), speed, heading, and timestamp. This could help in supporting maritime surveillance, such as detecting illegal fishing activities (e.g., identifying unauthorized operations), as well as behavior recognition and compliance monitoring (e.g., recognizing trawling and other regulated practices).

Another important application scenario is human motion trajectory prediction [1, 29, 68]. For instance, in the healthcare domain, trajectory information can be used to monitor whether an elderly person is currently in a safe condition [82]. This can help identify early signs of abnormal movement patterns (such as wandering, hesitation, or sudden stops) that may indicate a risk of falling or cognitive decline. By continuously analyzing motion trajectories in real-time, healthcare systems can provide timely alerts or interventions, thereby improving safety and rapid emergency response. Rudenko *et al.* [68] provided a comprehensive survey of methods for human motion trajectory prediction. They emphasize that machine learning, particularly neural network models, enables the extraction of latent behavioral patterns from noisy and context-rich environments.

Trajectory prediction has attracted widespread attention from both academia and industry [2, 3, 28, 59, 99]. Several widely used datasets have supported research in this field, including the GeoLife GPS Trajectories dataset [99], which contains fine-grained daily mobility trajectories of individuals; the Human Activity Recognition Using Smartphones dataset [3], which involves classifying human activity types (e.g., walking, sitting) based on motion signal trajectories; and the Boat dataset [2], which provides BeiDou-based fishing vessel trajectories annotated with operational status labels (e.g., trawling), supporting research in maritime behavior prediction and compliance monitoring. Notably, the Boat dataset was released in the Boat Fishing Vessel Operation Recognition Challenge hosted on the Alibaba Tianchi platform, which aimed to encourage the development of trajectory-based behavior classification algorithms for real-world maritime applications.

2.2 Test Input Prioritization for Deep Neural Networks

In the domain of deep neural networks, test input prioritization aims to identify and prioritize test cases that are more likely to be misclassified by the DNN model [15, 24, 47, 48]. Through labeling these potentially misclassified inputs earlier, the debugging process becomes more efficient, thereby improving the overall effectiveness of DNN testing. Moreover, when the labeling process is halted at an arbitrary point due to resource limitations, test prioritization could help maximize the benefit gained from limited human effort [24].

Existing mainstream test prioritization approaches for deep neural networks can be broadly categorized into coverage-based [57, 63, 78, 90], confidence-based [24, 84], and learning-based methods [13, 49, 50, 83]. Coverage-based approaches adapt traditional software testing techniques by leveraging neuron coverage or activation patterns to guide prioritization [57, 63]. Inspired by traditional software testing (e.g., code or MC/DC coverage), these methods define coverage criteria over neural activations, neuron interactions, or abstract decision behaviors. A foundational concept is neuron coverage, which measures the proportion of activated neurons and guides test input

generation or prioritization. This has been extended into finer-grained metrics like k-multisection and top-k coverage [57], enabling deeper assessment of model behavior.

Confidence-based approaches rank test inputs based on the prediction confidence output by the DNN model, under the assumption that lower-confidence predictions are more likely to be misclassified. Existing studies have proposed various confidence metrics. For instance, DeepGini [24] computes a Gini score from the predicted class probabilities to quantify uncertainty and prioritize inputs with higher uncertainty. Other popular uncertainty metrics include Prediction Confidence Score (PCS), entropy, and vanilla softmax probabilities [84]. Confidence-based methods have been proven to surpass coverage-based methods in both effectiveness and efficiency.

Learning-based approaches are currently the most popular test prioritization methods [50, 83], which train auxiliary models to automatically learn features from test inputs and use these features to predict the probability that the test inputs will be misclassified by the DL model. At present, the state-of-the-art method for tabular data (which can be adapted to the context of trajectory prediction) is MLPrior [15], which leverages the interpretable nature and structured features of classical models to design mutation generators and extract mutation features for test prioritization. MLPrior has been demonstrated to significantly outperform all existing test prioritization methods on tabular data.

3 PRELIMINARIES

In this section, we introduce the representative test input prioritization techniques that serve as compared methods in our evaluation. Specifically, we focus on two major categories: the state-of-the-art test prioritization MLPrior and a set of widely adopted confidence-based approaches that prioritize test cases based on model prediction uncertainty.

3.1 MLPrior

MLPrior [15] is a state-of-the-art test input prioritization approach specifically designed for classical machine learning models, and can be effectively adapted to test prioritization tasks in trajectory prediction scenarios since it utilizes tabular data. MLPrior is grounded in two core premises: 1) Test inputs that are more sensitive to small perturbations (either to the model or the input) are more likely to be misclassified, and 2) Test inputs located closer to the model's decision boundary are more likely to be misclassified.

Based on these premises, MLPrior generates three categories of features for each test input, including Model Mutation Features (MMF), Input Mutation Features (IMF), and Original Attribute Features (OAF).

- **Model Mutation Features (MMF)** capture how sensitive an input is to slight alterations in the model itself. This is achieved by mutating the original model (such as adjusting thresholds in decision trees, tweaking weights in logistic regression, or modifying classification thresholds in XGBoost) without retraining. If a test input's prediction changes frequently across these mutated models (i.e., it kills many mutants), it is considered more likely to be misclassified.
- **Input Mutation Features (IMF)** assess how sensitive an input is to perturbations in its own attributes. This is done by generating mutated variants of the input (e.g., by zeroing out individual features) and observing whether the model's predictions for the mutants differ from the original. Inputs with highly variable predictions are likely to be misclassified.
- **Original Attribute Features (OAF)** directly represent the original input features in numerical form and provide information about how close the input lies to the model's decision boundary. Prior studies suggest that inputs closer to this boundary are more likely to be misclassified.

MLPrior concatenates all three types of features into a final feature vector for each test input. This vector is then fed into a pre-trained learning-to-rank model (e.g., XGBoost), which outputs a misclassification likelihood score for each test. The final prioritization ranks all test inputs in descending order of this score, enabling developers to focus labeling and debugging efforts on the most fault-revealing test cases first. In summary, given a test input t, MLPrior computes its misclassification likelihood score as follows for the purpose of test prioritization.

$$\text{Score}(t) = \mathcal{R}\left(\phi_{\text{MMF}}(t) \,\|\phi_{\text{IMF}}(t)\|\, \phi_{\text{OAF}}(t)\right) \tag{1}$$

where $\phi_{\text{MMF}}(t) \in \mathbb{R}^{d_1}$ denotes the *model mutation features*, capturing the sensitivity of test t to slight perturbations in the model parameters; $\phi_{\text{IMF}}(t) \in \mathbb{R}^{d_2}$ denotes the input mutation features, measuring the prediction variability when the attributes of test t are perturbed; $\phi_{\text{OAF}}(t) \in \mathbb{R}^{d_3}$ represents the *original attribute features*, which is used to estimate its proximity to the decision boundary; $\|$ denotes vector concatenation; $\mathcal{R}(\cdot)$ is a pre-trained *learning-to-rank model* that maps the concatenated feature vector to a scalar score reflecting the likelihood of misclassification; $\text{Score}(t)$ is used to rank all test inputs in descending order, prioritizing those most likely to be misclassified.

3.2 Confidence-based Approaches

This section presents the confidence-based test input prioritization techniques evaluated in our research. Confidence-based methods rely on the model's prediction confidence to estimate the likelihood of misclassification. Test cases with lower prediction confidence are considered more likely to be misclassified.

- **DeepGini** [24] leverages the Gini score to quantify the uncertainty in the output distribution of a test input t. A higher Gini score indicates that the model is more uncertain about this test input, and it is therefore considered more likely to be misclassified. The Gini score is calculated as follows:

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

  where $p_i(t)$ is the softmax probability of input t being classified into label i, and N is the total number of labels.
- **Prediction-Confidence Score (PCS)** [84] measures the difference between the most confident and the second most confident class predictions. A smaller difference suggests the model is less certain about its prediction, indicating a higher probability of misclassification. The score is defined as:

$$P(t) = l_k(t) - l_j(t) \tag{3}$$

  where $l_k(t)$ and $l_j(t)$ denote the highest and second-highest softmax probabilities, respectively.
- **Vanilla Softmax** [84] quantifies prediction uncertainty by comparing the maximum softmax score to the ideal value of 1. A lower maximum value implies greater uncertainty. The misclassification score is calculated by:

$$V(t) = 1 - \max_{c=1}^{C} l_c(t) \tag{4}$$

  where $l_c(t)$ is the softmax probability for class c, and C is the total number of classes.
- **Entropy** [84] assesses the uncertainty of predictions by computing the Entropy of the softmax distribution. Higher entropy values indicate more evenly distributed probabilities, which typically correlate with a higher chance of misclassification. The entropy-based score is defined as:

$$H(t) = -\sum_{i=1}^{N} p_i(t) \cdot \log p_i(t) \tag{5}$$

where $p_i(t)$ is the probability of class i for test input t.

4 APPROACH

In this section, we present the workflow and core rationale of our proposed test prioritization approach, TrajectoryTest, which is specifically designed for trajectory prediction tasks. The design of TrajectoryTest considers the limitations of the state-of-the-art learning-based method MLPrior. ***Limitation 1:*** MLPrior could suffer from training data imbalance when applied to high-accuracy models. This is because the training dataset for MLPrior's ranking model is labeled according to whether a sample is misclassified (labeled as 1) or correctly classified (labeled as 0). If only a small number of test inputs are misclassified (i.e., the model has very high accuracy), the proportion of positive samples becomes extremely low, leading to severe class imbalance. Such imbalance can greatly restrict the learning effectiveness of the ranking model, thereby reducing the prioritization performance of learning-based methods; ***Limitation 2:*** Methods like MLPrior rely on rich and diverse input features, whereas trajectory datasets typically provide only limited spatiotemporal attributes, which are not sufficiently rich. Table 8 presents a sample data point in a trajectory. As shown, the features of a single data point consist only of its latitude, longitude, and time, with very few raw attributes included.

To overcome the aforementioned limitations, TrajectoryTest introduces the following novelty for test prioritization:

- **Adaptive strategy based on error ratio** To address the first limitation, TrajectoryTest adopts an adaptive strategy guided by the error ratio. Instead of blindly applying a learning-based ranking model, TrajectoryTest first evaluates the model's error ratio. If misclassifications are too few, it avoids class imbalance by switching to uncertainty-based prioritization. Otherwise, it proceeds with a learning-to-rank approach.
- **Trajectory-aware prioritization** To address the second limitation, TrajectoryTest leverages trajectory-specific information for test prioritization. It enriches the ranking model with spatiotemporal trajectory features (e.g., dwell time, visit count, heading, speed, and statistical information), thereby enabling more accurate prioritization tailored to trajectory datasets.

Figure 1 illustrates the overall workflow of TrajectoryTest, which is divided into the following three main steps. A brief description is provided below, while the detailed procedures of each step can be found in Section 4.1 to Section 4.4.

❶ **Step 1: Compute Error Ratio** The trajectory dataset (training + test sets) is fed into the prediction model to compute the error ratio on the training set. This ratio indicates whether sufficient misclassified samples exist to train a reliable ranking model. Specifically, it computes the proportion of misclassified samples by the target model within the training dataset. A lower error ratio indicates fewer misclassified samples, and thus, higher model accuracy. This can result in an imbalanced training set for building the ranking model for learning-based methods. This is because the training labels for the ranking model are based on whether a test case is "misclassified" (labeled as 1) or "correctly classified" (labeled as 0). If only a few samples are misclassified, the number of "misclassified" cases will be too few, leading to class imbalance. In such cases, we avoid the learning-based approach and instead proceed to Step 2, performing uncertainty-based analysis using the model's predictive uncertainty to guide test prioritization. When the error ratio is higher, it indicates that the training data is relatively balanced. In this case, we proceed to Step 3, where we leverage trajectory information for test prioritization.

❷ **Step 2: Confidence-Based Prioritization (if error ratio < threshold)** When the error ratio is low (i.e., too few misclassifications), a learning-based model would suffer from class imbalance. In this case, we adopt an uncertainty-based strategy: each test input is scored by the model's predictive uncertainty, and tests are ranked in descending order of uncertainty.

❸ **Step 3: Trajectory-Aware Prioritization (if error ratio ≥ threshold)** When enough misclassifications exist, we extract trajectory-specific spatiotemporal features (e.g., dwell time, visit count, heading, speed) and combine them with MLPrior-derived features. A learning-to-rank model is then trained on this feature set to predict misclassification likelihood scores, which are used to rank test inputs.

To further clarify the core mechanism of TrajectoryTest, we now provide a concrete illustrative example demonstrating how the method works in practice.

Problem Setup. Given a trajectory dataset composed of a labeled training set and an unlabeled test set, the objective of TrajectoryTest is to prioritize trajectory tests that are more likely to be misclassified by the machine learning model M under evaluation.

(1) **Compute Error Ratio.** We first feed the training trajectories $(x_1, x_2, \ldots, x_t)$ into model M and compare the predicted outputs with the ground truth labels. Then we computed the Error Ratio, which is the proportion of misclassified samples in the training set. For example, suppose the model misclassifies 10% of the training samples; then the error ratio is 0.10.

(2) **Compare with Threshold.** The error ratio is then compared to a predefined threshold (default: 0.05). This determines whether TrajectoryTest will follow a learning-based strategy or a confidence-based one: If the error ratio is lower than the threshold (e.g., 0.02), the training set is considered too imbalanced for learning. TrajectoryTest will switch to a confidence-based strategy. If the error ratio is higher than the threshold (e.g., 0.10), the training data is sufficiently balanced, and a learning-based strategy will be applied.

(3) **Branch A: Confidence-Based Ranking** Suppose the error ratio is 0.02. In this case, TrajectoryTest passes each test trajectory t through model M to obtain a prediction probability vector, denoted as $[p_1, p_2, \ldots, p_K]$, where K is the number of classes. Each p_i represents the predicted probability that trajectory t belongs to class i. We then apply uncertainty metrics to quantify how uncertain the model is for each test input. The test trajectories are ranked based on their uncertainty scores, with those having higher uncertainty being prioritized higher.

(4) **Branch B: Learning-Based Ranking** Suppose the error ratio is 0.10. In this case, TrajectoryTest proceeds with a learning-based ranking strategy. For each training trajectory, trajectory-aware spatiotemporal features (such as dwell time, speed, heading, visit count, and statistical summaries) are extracted and combined with MLPrior-derived features to form enriched representations $(r_1, r_2, \ldots, r_n)$. Each test is then labeled as 1 if it was misclassified by model M, or 0 otherwise. These labeled feature vectors are used to train a learning-to-rank model. After the model is well-trained, for the test set $(t_1, t_2, \ldots, t_m)$, the same feature extraction process is applied, and the resulting representations are fed into the trained ranking model to obtain misprediction scores $(p_1, p_2, \ldots, p_m)$. Finally, the test trajectories are ranked in descending order of these scores, with higher-ranked trajectories being more likely to be misclassified. For example, suppose the ranking model outputs scores $p_1 = 0.85$, $p_2 = 0.42$, and $p_3 = 0.67$ for test trajectories t_1, t_2, and t_3, respectively. Then the ranked order would be $t_1 \succ t_3 \succ t_2$, indicating that t_1 is most likely to be misclassified, followed by t_3, and finally t_2.

In the following sections, we first provide a detailed description of each step in the workflow, and then explain how to apply TrajectoryTest.

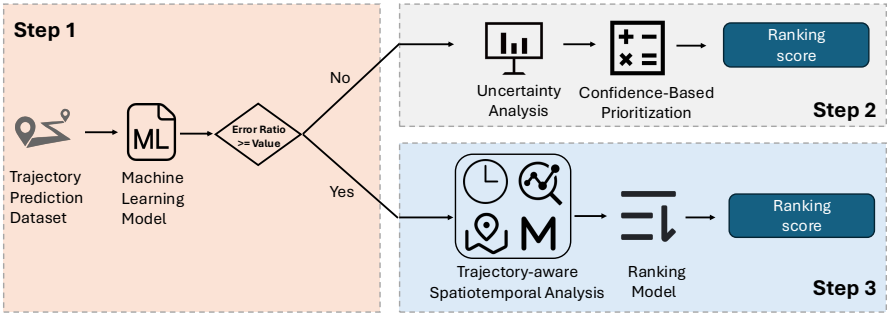


Fig. 1. Overview of TrajectoryTest

4.1 Step 1: Compute the Error Ratio

In this step, we input the trajectory dataset, which includes both the training set and the test set, into the trajectory prediction model under evaluation. The purpose is to compute the Error Ratio on the training set, defined as the proportion of samples misclassified by the model. Formula 6 shows the detailed computation process.

$$ER = \frac{N_{\text{err}}}{N_{\text{train}}} \tag{6}$$

where ER denotes the Error Ratio, N_{err} is the number of misclassified samples in the training set. N_{train} is the total number of samples in the training set.

This ratio is used to estimate the feasibility of constructing a balanced training set for the learning-to-rank model employed in learning-based methods. In learning-based approaches, if this error ratio is too low, the constructed training set becomes more imbalanced, making it less effective. The main reason is: the learning-based ranking model requires training labels that indicate whether each test case is misclassified (labeled as 1) or correctly classified (labeled as 0). When only a small number of samples are misclassified, the resulting imbalance between positive and negative labels leads to unreliable training for the ranking model. In such cases, we proceed to Step 2, where we perform uncertainty-based analysis. Here, the model's predictive uncertainty is used to guide test prioritization, allowing us to identify potentially misclassified inputs even when the model exhibits high accuracy. Otherwise, we proceed to Step 3, where we apply a trajectory-specific learning-based method for test prioritization.

In the default TrajectoryTest, we set the error ratio threshold to 0.05 to determine whether to adopt the confidence-based or learning-based strategy. This threshold is chosen based on prior empirical findings. Zheng *et al.* [98] demonstrated that model performance degrades when trained on datasets with imbalanced distributions between positive and negative samples. Moreover, the greater the imbalance, the more significant the performance degradation. The selected threshold of 0.05 falls within the range identified in their study as representing a significantly imbalanced dataset. Their experimental results further confirm that, within this range, models trained on such imbalanced data suffer noticeable performance drops. Moreover, in RQ5 (cf. Section 6.5), we further experimented with different threshold values and examined how varying the threshold affects the effectiveness of TrajectoryTest.

4.2 Step 2: Confidence-Based Prioritization

If the error ratio computed in Step 1 is below a predefined threshold (indicating a low number of misclassifications in the training set), we skip learning-based ranking due to potential class

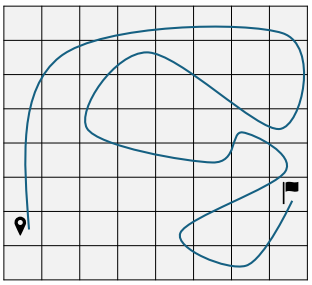


Fig. 2. Trajectory-specific information

imbalance issues. Instead, we perform confidence-based test input prioritization by calculating the model's predictive uncertainty on the test set.

These scores quantify how uncertain the model is about its predictions. Test cases with higher uncertainty values are considered more likely to be misclassified. We rank the test inputs in descending order of uncertainty score and prioritize those at the top for inspection or labeling. This approach ensures that we focus on potentially misclassified predictions even in highly accurate models, where learning a ranking model would be unreliable due to insufficient negative (i.e., misclassified) training examples. This process can be represented by the following Formula 7.

$$\text{Rank} = \text{argsort}_{x_i \in \mathcal{X}_{\text{test}}} \left(U\left(x_i\right) \right) \tag{7}$$

where $\mathcal{X}_{\text{test}}$ denotes the set of test inputs. $x_i \in \mathcal{X}_{\text{test}}$ represents an individual test input. $U(x_i)$ denotes the predictive uncertainty of the model for input x_i. argsort$(\cdot)$ returns the indices of inputs sorted by the given uncertainty score. The sorting is performed in descending order so that test inputs with higher uncertainty are prioritized. Rank is the resulting prioritized list of test inputs, ordered by decreasing uncertainty scores.

4.3 Step 3: Prioritization with trajectory information

If the error ratio is above the predefined threshold, it indicates that the training set contains a sufficient number of misclassified examples to train a reliable learning-to-rank model. In this case, we proceed with a trajectory-aware prioritization strategy. Specifically, we extract two categories of information for each sample:

- **Trajectory-specific information** These features encode trajectory-level spatial and motion patterns (e.g., dwell time, visit count, heading, and speed), which reflect the underlying behavioral characteristics of the trajectory. First, we discretize the trajectory path (e.g., GPS latitude and longitude) into an $m \times n$ grid of cells, as illustrated in Figure 2. For each trajectory, we compute the following features within each cell: dwell time, visit count, heading/orientation, and speed. The rationale for extracting these features is as follows:
  (1) **Dwell Time** reflects regions where the trajectory "pauses" spatially. Different types of trajectories often exhibit significant differences in dwell time at specific locations. As a temporal signal of spatial behavior patterns, dwell time helps reveal underlying activity regularities.
  (2) **Visit Count** records how many times a cell is traversed, highlighting spatial coverage. Visit count provides the spatial density distribution of a trajectory, which complements dwell time.
  (3) **Heading/Orientation** describes the movement direction distribution across space, which characterizes motion patterns. Orientation could help reveal the structural properties of the trajectory path.

(4) **Speed** characterizes motion state. Different types of trajectories usually show distinct speed distributions.
(5) **Statistical Information** summarizes trajectory-specific statistical characteristics. Based on the grid representation, we compute: 1) **Mean**, which describes the average position of a trajectory or the general region it tends to occupy; 2) **Variance/Standard Deviation**, which reflect the degree of dispersion (i.e., whether the movement is spatially concentrated or widely spread); 3) **Quartiles**, which capture the typical distribution range of the trajectory and are robust against outliers; 4) **Minimum/Maximum**, which indicate the boundary range of the trajectory, such as the furthest point reached. Different categories of trajectories exhibit clear differences in their statistical characteristics. For instance, walking and running show different average dwell times within each cell, with walking trajectories typically having significantly longer mean dwell times than running trajectories.

- **MLPrior-derived information** are generated based on the state-of-the-art test prioritization approach MLPrior. We provide a detailed description of the approach MLPrior in Section 3.1, including the types of features it generates and the information they encapsulate. Then, for each test input, we concatenate the trajectory-specific features described above with the MLPrior-derived features to construct a final representation.

Finally, we train a learning-to-rank model, as described in Section 4.4. After training, given a test input, the model predicts a misclassification likelihood score, indicating the probability that the target model will misclassify it. These scores are then used to rank all test samples in the test set in descending order, thereby enabling effective prioritization of the tests.

4.4 Usage of TrajectoryTest

In this section, we present the usage of TrajectoryTest. Given an unordered, unlabeled test set $T = \{t_1, t_2, \ldots, t_n\}$, a training set R, and a trained model M to be evaluated, TrajectoryTest outputs a ranked list of test inputs based on their likelihood of being misclassified. The process consists of the following steps:

① **Error ratio computation** Compute the model M's error ratio on the training set, where the detailed formula for computing this value can be found in Section 4.1.
② **Confidence-based prioritization** If the error ratio is smaller than the threshold, it indicates that there are insufficient misclassified tests. In this case, we switch to the confidence-based process, which calculates the model's predictive uncertainty on the test set for test prioritization.
③ **Learning-based prioritization** If the error ratio is greater than the threshold, we switch to the learning-based method driven by trajectory information, which is further divided into the following four sub-steps:
  - **Feature Generation:** For each input trajectory, we generate two categories of features: *1) Trajectory-based features*, which encode spatiotemporal patterns such as dwell time, visit count, heading, speed, and statistical information; and *2) MLPrior-derived features*, generated by the state-of-the-art MLPrior method. These features form the foundation for subsequent training set construction.
  - **Training Set Construction:** For each training input $r_i \in R$, we concatenate the trajectory-based and MLPrior-derived features into a final feature vector V_i. We then input r_i into M to obtain its prediction L_i. By comparing L_i with the ground truth, we assign a label (1 if misclassified, 0 otherwise), thus forming the labeled training set R'.
  - **Ranking Model Training:** Using the constructed set R', we train a learning-to-rank model. After training, the model can estimate a misclassification likelihood score for any test input based on its combined feature vector.

– **Test Input Prioritization:** Finally, we apply the trained ranking model to the test set T. Each test input receives a misclassification likelihood score, and the test inputs are prioritized in descending order of these scores. The resulting ranked list highlights the most potentially misclassified inputs.

5 STUDY DESIGN

In this section, we present a detailed exposition of our experimental study design. Section 5.1 outlines the four research questions that form the foundation of our investigation, focusing on evaluating existing test prioritization methods and the proposed TrajectoryTest in trajectory prediction tasks. In Section 5.2, we describe the datasets and models employed in our experiments. Section 5.3 details the construction of noisy trajectory data. Section 5.4 defines the evaluation metrics to measure test prioritization effectiveness. Finally, Section 5.5 specifies the implementation and configuration details of the experimental setup.

5.1 Research questions

Our experimental evaluation answers the research questions below.

- **RQ1: How does the proposed method, TrajectoryTest, perform compared to existing test prioritization techniques on natural trajectory prediction data?**
  Trajectory prediction plays a critical role in several safety-sensitive domains [68, 81], where accurate and reliable model behavior is essential. Given the high cost of labeling and the potential consequences of misprediction, test prioritization has emerged as a crucial method to improve testing efficiency and reduce the labeling effort. However, the existing state-of-the-art test prioritization method, MLPrior, exhibits several limitations when applied to the trajectory prediction scenario: it relies heavily on rich input features, which are often unavailable in trajectory data, and its effectiveness could be significantly affected when the training set used to build the ranking model is imbalanced. Therefore, we propose TrajectoryTest, a method specifically designed for the trajectory prediction scenario. In this research question, we aim to empirically validate the limitations of MLPrior and investigate whether the proposed method, TrajectoryTest, can outperform the state-of-the-art approach MLPrior in the context of trajectory prediction.
- **RQ2: How does TrajectoryTest perform in the presence of noise, compared to existing test prioritization techniques?**
  Real-world trajectory prediction data could contain noise due to sensor drift, signal loss, or environmental interference. To more comprehensively evaluate the effectiveness of TrajectoryTest relative to existing test prioritization methods, we simulate such conditions by injecting controlled noise into the input data to generate noisy datasets. We then compare TrajectoryTest with the state-of-the-art method MLPrior, confidence-based approaches, and the random selection baseline on these noisy datasets, aiming to determine whether TrajectoryTest can outperform these methods under noisy conditions.
- **RQ3: Does each component of TrajectoryTest contribute to its effectiveness?**
  The workflow of TrajectoryTest integrates several components. This research question investigates whether each component contributes to the overall effectiveness of TrajectoryTest through an ablation study.
- **RQ4: To what extent does the adaptive switching mechanism contribute to TrajectoryTest?**
  TrajectoryTest employs an adaptive switching mechanism that dynamically chooses between confidence-based and learning-based prioritization strategies based on the error ratio (i.e., the

proportion of misclassified samples by the target model in the training dataset). This mechanism is designed to ensure more stable and effective performance by avoiding unreliable learning from highly imbalanced data. This research question aims to quantify the contribution of this adaptive switching strategy by comparing the full adaptive version of TrajectoryTest with a non-adaptive variant that always applies the learning-based prioritization strategy regardless of the error ratio.

- **RQ5: What is the impact of the error ratio threshold on the effectiveness of TrajectoryTest?**
  The adaptive switching mechanism in TrajectoryTest relies on an error ratio threshold to determine whether to use confidence-based or trajectory-aware learning-based prioritization. This research question investigates how sensitive TrajectoryTest is to different threshold values. We varied the threshold while keeping all other components unchanged to isolate its effect and compare the effectiveness of TrajectoryTest under different threshold settings.

5.2 Models and Datasets

In our study, we empirically evaluate existing test prioritization methods in the context of trajectory prediction and further propose new strategies tailored for this domain. To support the investigation, we utilize three popular datasets: GeoLife GPS Trajectories [99], Boat [2], and Human Activity Recognition Using Smartphones (HAR) [3], as well as seven machine learning/deep learning models. Table 1 presents the detailed information of the datasets, including the total number of data points (Data point count), the number of trajectories (Trajectory count), the models used (Models), and the data types (where Original refers to noise-free natural data, and Noisy refers to data with added noise).

We selected these datasets because: 1) they are widely utilized in both academia and industry [25, 61, 66], and 2) they reflect diverse challenges within trajectory-based tasks. GeoLife provides real-world GPS traces from users across different transportation modes. The Boat dataset focuses on maritime operation behavior recognition using vessel movement data. The HAR dataset centers on human activity recognition, classifying daily human activities using smartphone sensor data.

In our experiments, we utilize a total of 42 subjects: 21 subjects from the original datasets (3 natural datasets × 7 models) and 21 subjects from the noisy datasets (3 noisy datasets × 7 models), for evaluating the effectiveness of TrajectoryTest, the state-of-the-art approach MLPrior, confidence-based methods and the baseline method random selection.

5.2.1 Datasets.

- **Boat** [2] The Boat dataset (released by the Alibaba Cloud Tianchi platform) is designed for maritime operation behavior classification based on positioning data. It involves the recognition of fishing operation types such as trawling, purse seining, and gillnetting. The dataset includes over 2,692,638 data points from fishing vessels, with each record capturing vessel ID, coordinates (x, y), timestamp, and operation label.
- **GeoLife** [99] The task of the GeoLife GPS Trajectories dataset is to classify user transportation modes based on GPS trajectory data. Example labels for transportation mode classification include walk, bike, bus, car, subway, train, airplane, boat, run, motorcycle, and taxi. This dataset is widely used in research areas including trajectory mining, user movement prediction, and activity recognition.
- **Activity** [3] The task of the Human Activity Recognition Using Smartphones dataset is to classify six types of daily human activities using smartphone sensor data. The dataset contains recordings from 30 volunteers aged 19–48 who performed activities while wearing a Samsung Galaxy S II smartphone on their waist. The dataset is commonly used in research involving multivariate

Table 1. Trajectory prediction datasets and models

ID	Datasets	Data point count	Trajectory count	Models	Data Type
1	Boat	2,692,638	7,000	DNN	Original, Noisy
2	Boat	2,692,638	7,000	Tree	Original, Noisy
3	Boat	2,692,638	7,000	KNN	Original, Noisy
4	Boat	2,692,638	7,000	LR	Original, Noisy
5	Boat	2,692,638	7,000	RF	Original, Noisy
6	Boat	2,692,638	7,000	XGB	Original, Noisy
7	Boat	2,692,638	7,000	LGBM	Original, Noisy
8	GeoLife	5,509,014	9,615	DNN	Original, Noisy
9	GeoLife	5,509,014	9,615	Tree	Original, Noisy
10	GeoLife	5,509,014	9,615	KNN	Original, Noisy
11	GeoLife	5,509,014	9,615	LR	Original, Noisy
12	GeoLife	5,509,014	9,615	RF	Original, Noisy
13	GeoLife	5,509,014	9,615	XGB	Original, Noisy
14	GeoLife	5,509,014	9,615	LGBM	Original, Noisy
15	Activity	5,777,739	10,299	DNN	Original, Noisy
16	Activity	5,777,739	10,299	Tree	Original, Noisy
17	Activity	5,777,739	10,299	KNN	Original, Noisy
18	Activity	5,777,739	10,299	LR	Original, Noisy
19	Activity	5,777,739	10,299	RF	Original, Noisy
20	Activity	5,777,739	10,299	XGB	Original, Noisy
21	Activity	5,777,739	10,299	LGBM	Original, Noisy

time-series classification. Example labels include walking, walking upstairs, walking downstairs, sitting, standing, and lying.

5.2.2 *Models.*

To evaluated the effectiveness of TrajectoryTest and other test prioritization methods, we adopted the following 7 models, Decision Tree (DT) [75], K-Nearest Neighbors (KNN) [42], Logistic Regression (LR) [44], Random Forest (RF) [30], XGBoost(XGB) [11], LightGBM(LGBM) [22], and Deep Neural Network (DNN) [39]. These models were selected based on two key considerations: 1) they have been widely used in trajectory-related prediction tasks due to their strong performance and ease of implementation [56, 65, 91, 94]; and 2) they represent a diverse set of model families (including linear, instance-based, tree-based, ensemble, and deep learning methods) with the aim of enabling a more comprehensive evaluation.

- **Decision Tree (DT)** Decision Tree works by recursively partitioning the feature space into subsets based on feature values that maximize information gain or minimize impurity (e.g., Gini index or entropy). At each internal node, the algorithm selects the best feature and threshold to split the data such that the resulting child nodes are as pure as possible. This process continues until a stopping criterion is met (e.g., maximum depth or minimum number of samples per leaf).
- **K-Nearest Neighbors (KNN)** K-Nearest Neighbors is an instance-based model that predicts a sample's label by aggregating the labels of its k closest neighbors in the feature space. Distance metrics such as Euclidean, Manhattan, or cosine similarity are typically used to determine proximity. The model makes no assumptions about the data distribution and requires no training phase.
- **Logistic Regression (LR)** Logistic Regression is a linear classifier that models the probability of a binary or multi-class outcome using the logistic (sigmoid) function. It estimates the parameters by maximizing the likelihood function under the assumption that the log-odds of the outcome is a linear combination of the input features.
- **Random Forest (RF)** Random Forest is an ensemble method that constructs multiple decision trees on bootstrapped subsets of data and aggregates their predictions (via majority vote for

classification or averaging for regression) to reduce variance. During tree construction, a random subset of features is considered at each split, introducing additional diversity among trees.

- **XGBoost(XGB)** XGBoost is a highly optimized gradient boosting framework that builds additive decision trees in a stage-wise manner. At each iteration, a new tree is trained to minimize the loss function by fitting the negative gradient (i.e., the residuals) of the current model. XGBoost incorporates advanced features such as second-order optimization (using both first and second derivatives), regularization (to control model complexity), and efficient handling of missing values and sparsity-aware split finding.
- **LightGBM(LGBM)** LightGBM is a gradient boosting framework optimized for efficiency and scalability. It uses histogram-based feature binning to discretize continuous features into buckets, significantly reducing memory usage and computation cost. Unlike level-wise tree growth (as in traditional boosting), LightGBM adopts a leaf-wise growth strategy that splits the leaf with the highest loss reduction, resulting in deeper trees and better accuracy.
- **Deep Neural Network (DNN)** Deep Neural Network is a multi-layer architecture that learns hierarchical representations through non-linear transformations. Each layer applies a linear transformation to its input (via learned weights and biases), followed by a non-linear activation function (e.g., ReLU, tanh). Through backpropagation and gradient-based optimization (e.g., SGD or Adam), the network updates its parameters to minimize a predefined loss function.

5.3 Trajectory Noise

In real-world applications, trajectory data can contain noise due to sensor drift, signal interference, or environmental uncertainty [6]. Such noise can significantly impact model behavior. In RQ2 (cf. Section 6.2), we investigate the effectiveness of existing test prioritization techniques. The noisy data is created based on the three datasets in Section 5.2.1 (i.e., GeoLife GPS Trajectories, Boat and Human Activity Recognition Using Smartphones). Following existing research [15], we generate noisy test data by introducing slight random perturbations to a portion of the samples in the datasets. We then create a mixed noisy dataset by combining 50% noisy data with 50% natural data.

5.4 Measurements

To assess the effectiveness of existing test prioritization methods and our proposed method TrajectoryTest, we adopt two metrics commonly used in the literature [24]: the Average Percentage of Fault Detection (APFD) and the Percentage of Faults Detected (PFD).

- **Average Percentage of Fault Detection (APFD)** APFD [89] is a well-established measure for quantifying the effectiveness of test prioritization methods. It evaluates how fast a test prioritization method can detect misclassified tests. A larger APFD value reflects faster fault detection rates.

$$APFD = 1 - \frac{\sum_{i=1}^{k} o_i}{kn} + \frac{1}{2n} \tag{8}$$

where n is the total number of test cases. k is the number of misclassified samples in the dataset. o_i is the position of the i-th misclassified input within the ranked list generated by the prioritization method. Thus, $\sum_{i=1}^{k} o_i$ captures the cumulative position of misclassified tests. Following the setup in prior work [24], we normalize the APFD scores to fall within the range [0,1] for consistent comparison.

- **Percentage of Faults Detected (PFD)**

PFD quantifies the proportion of misclassified inputs that are detected within a given portion of the prioritized test set. It provides a direct view of the fault detection rate at different prioritization depths. The metric is defined as follows:

$$PFD = \frac{N_d}{N} \tag{9}$$

where N_d is the number of detected misclassified inputs among the top-ranked test cases, and N is the total number of misclassified samples. In our evaluation, we report **PFD-n**, where n refers to the ratio of tests executed. This allows us to assess how effectively each method, including TrajectoryTest, is able to expose faults when specific ratio of tests are executed.

- **Normalized Average Percentage of Fault Detection (NAPFD)**
  NAPFD is a normalized variant of the APFD metric. While APFD already reflects how quickly misclassified tests are detected, NAPFD ensures fair comparison across models and datasets by accounting for differences in the density of faults. A larger NAPFD value indicates faster and more consistent detection performance relative to the total number of faults. The metric is defined as follows:

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

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

5.5 Implementation and Configuration

In our experiments, we employed PyTorch 1.13.1+cu117 and TensorFlow 2.2.0 as the deep learning frameworks, XGBoost 1.5.0, and scikit-learn 1.0.2 as the machine learning frameworks. The threshold of the error ratio in the TrajectoryTest workflow is set to 0.05. Under this threshold, across all original datasets (a total of 21 subjects composed of 7 models and 3 datasets), 5 subjects were assigned to the confidence-based branch, while 16 subjects were assigned to the trajectory-aware learning-based branch. For data processing, we employed Numpy 1.21.6 and Pandas 1.3.5. Our experimental environment was based on a high-performance computing cluster. Each cluster node was equipped with a 2.6 GHz Intel Xeon Gold 6132 CPU and an NVIDIA Tesla V100 16 GB SXM2 GPU. For additional data processing and analysis tasks, we used a MacBook Pro laptop running macOS Sequoia Version 15.5, equipped with an Intel Core i9 CPU and 64 GB of RAM.

6 RESULTS AND ANALYSIS

In this section, we conduct corresponding experiments to answer each of our research questions in Section 5.1. Specifically, for RQ1, we evaluate the effectiveness of existing test prioritization approaches (including the state-of-the-art approach and other methods) on natural test inputs in trajectory prediction tasks. For RQ2, we investigate their effectiveness under noisy conditions. For RQ3, we assess the effectiveness of our proposed strategy, TrajectoryTest, in comparison with existing methods. For RQ4, we perform an ablation study to analyze the contribution of each component in TrajectoryTest.

6.1 RQ1: Comparison of TrajectoryTest and Existing Test Prioritization Techniques on Natural Test Cases

Objectives: We aim to validate the limitations of existing test prioritization methods in the context of trajectory prediction through empirical evaluation, and to demonstrate whether TrajectoryTest, our newly proposed method specifically designed for the trajectory prediction domain, can outperform existing approaches.

Experimental design: We designed the following experiments to validate the limitations of existing methods and to demonstrate the effectiveness of our approach.

1) Effectiveness Evaluation We conducted a comparative evaluation of TrajectoryTest against several existing test prioritization methods, including the state-of-the-art MLPrior, four representative confidence-based techniques (DeepGini, Entropy, PCS, and VanillaSM), and the baseline method based on random selection. All methods were evaluated on a consistent experimental setup, involving seven different models across three trajectory prediction datasets, resulting in a total of 21 subjects. To measure effectiveness, we adopted three widely used metrics from prior work: Average Percentage of Fault Detection (APFD), Percentage of Fault Detected (PFD), and Normalized Average Percentage of Fault Detection (NAPFD). Full details regarding the computation of these metrics are presented in Section 5.4.

2) Statistical Analysis To ensure the statistical validity of our experiments, we conducted a statistical analysis by repeating all experiments ten times [14] and reporting the average results. We adopted two classical statistical measures: 1) the p-value and 2) the effect size, to demonstrate that the effectiveness differences between TrajectoryTest and the baseline methods are statistically significant. Below, we provide a detailed explanation of their computation and underlying principles.

- **P-value:** We employed the Wilcoxon signed-rank test [85], a non-parametric statistical hypothesis test used to compare paired samples. It evaluates whether the median of the differences between paired observations is significantly different from zero and is particularly appropriate when the data do not necessarily follow a normal distribution. A result is considered statistically significant when the p-value is less than 10^{-5} [58].
  In accordance with the standard formulation of the test, we define the hypotheses as follows:
  - **Null hypothesis (H_0):** The median difference between the paired results is zero, indicating no significant performance difference between our method and the baselines.
  - **Alternative hypothesis (H_1):** The median of the differences between paired results is not zero, suggesting a statistically significant difference in performance.

  We evaluate these hypotheses based on the p-values obtained from the test. In addition, we report Cliff's delta to quantify the effect size, which is presented as follows.
- **Effect size:** To quantify the magnitude of the differences, we used Cliff's delta [60], a non-parametric effect size measure suitable for ordinal or non-normally distributed data. Cliff's delta values range from -1 to +1, where 0 indicates no difference, and values closer to -1 or +1 indicate stronger effects. The interpretation thresholds are: $|\delta| < 0.15$ (negligible), $|\delta| < 0.33$ (small), $|\delta| < 0.47$ (medium), and $|\delta| \geq 0.47$ (large).

3) Efficiency analysis In addition to evaluating the effectiveness of TrajectoryTest and the existing approaches, we further assessed their efficiency by measuring the execution time. We aim to provide insights into the computational cost of TrajectoryTest and evaluate whether TrajectoryTest can be feasibly adopted in real-world trajectory prediction scenarios, where both effectiveness and efficiency are critical.

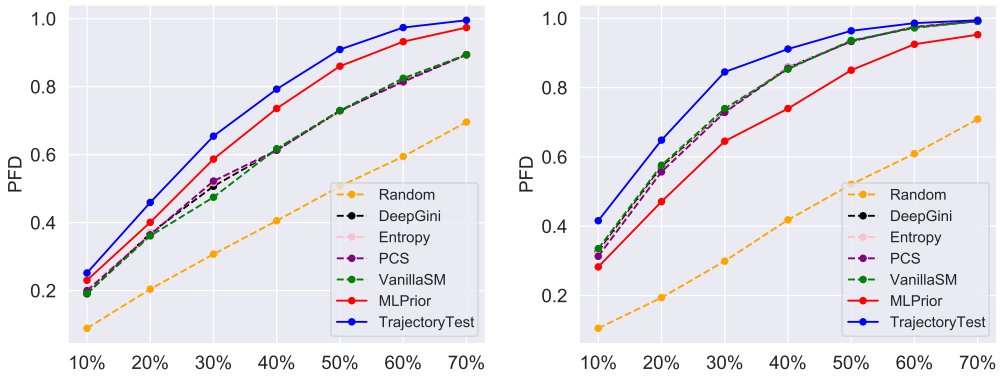
Results: The experimental results of RQ1 are presented in Table 2, Table 3, Table 4, Table 5, Figure 3 and Figure 4. The results highlight two key findings: 1) In the trajectory prediction scenario, the effectiveness of the state-of-the-art method MLPrior is relatively unstable and suboptimal. 2) Our

Table 2. Effectiveness comparison between TrajectoryTest and existing test prioritization strategies in terms of APFD

Data	Model	Approach						
		Random	DeepGini	Entropy	PCS	VanillaSM	MLPrior	**TrajectoryTest**
Activity	DNN	0.479	0.798	0.784	0.806	0.803	0.911	0.930
	Tree	0.513	0.740	0.739	0.739	0.739	0.861	0.905
	KNN	0.527	0.901	0.901	0.901	0.902	0.811	0.901
	LR	0.543	0.945	0.944	0.945	0.946	0.861	0.945
	RF	0.502	0.936	0.914	0.947	0.944	0.712	0.948
	XGB	0.532	0.969	0.968	0.970	0.969	0.762	0.969
	LGBM	0.455	0.965	0.965	0.966	0.965	0.705	0.965
Boat	DNN	0.499	0.676	0.675	0.676	0.676	0.783	0.786
	Tree	0.496	0.647	0.647	0.647	0.648	0.752	0.772
	KNN	0.505	0.678	0.678	0.677	0.678	0.741	0.766
	LR	0.504	0.655	0.651	0.657	0.656	0.722	0.755
	RF	0.522	0.795	0.793	0.799	0.801	0.766	0.799
	XGB	0.512	0.792	0.793	0.789	0.791	0.749	0.801
	LGBM	0.480	0.792	0.793	0.791	0.792	0.734	0.799
Geolife	DNN	0.496	0.508	0.495	0.533	0.524	0.617	0.626
	Tree	0.498	0.629	0.629	0.627	0.629	0.650	0.678
	KNN	0.497	0.563	0.562	0.551	0.557	0.595	0.626
	LR	0.502	0.531	0.483	0.529	0.532	0.618	0.626
	RF	0.518	0.649	0.636	0.663	0.657	0.599	0.689
	XGB	0.501	0.656	0.656	0.652	0.657	0.601	0.675
	LGBM	0.499	0.655	0.653	0.653	0.656	0.612	0.669

Table 3. Overall comparison results in terms of APFD

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.504	56.94
DeepGini	0	0.737	7.33
Entropy	0	0.732	8.06
PCS	2	0.738	7.18
VanillaSM	3	0.739	7.03
MLPrior	0	0.722	9.56
TrajectoryTest	16	0.791	-



(a) Boat, LR (b) Boat, XGB

Fig. 3. Effectiveness comparison on the dataset Boat with the model LR/XGB. X-Axis: the percentage of prioritized tests; Y-Axis: the percentage of detected misclassified tests.

proposed method, TrajectoryTest, significantly outperforms the state-of-the-art test prioritization method MLPrior, the confidence-based test prioritization methods, and the baseline method random

Table 4. Effectiveness comparison between TrajectoryTest and existing test prioritization strategies in terms of PFD

Data	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
Activity	Random	0.118	0.197	0.288	0.399	0.483	0.620	0.691
	DeepGini	0.725	0.869	0.906	0.937	0.952	0.964	0.977
	Entropy	0.680	0.863	0.902	0.934	0.951	0.964	0.978
	PCS	0.733	0.868	0.908	0.938	0.950	0.965	0.977
	VanillaSM	0.731	0.868	0.907	0.937	0.951	0.964	0.977
	MLPrior	0.491	0.684	0.778	0.836	0.879	0.912	0.938
	TrajectoryTest	0.825	0.964	0.977	0.988	0.991	0.992	0.997
Boat	Random	0.097	0.201	0.305	0.403	0.511	0.612	0.712
	DeepGini	0.249	0.455	0.624	0.747	0.830	0.890	0.931
	Entropy	0.250	0.453	0.620	0.744	0.830	0.891	0.930
	PCS	0.246	0.458	0.628	0.747	0.833	0.888	0.931
	VanillaSM	0.247	0.458	0.626	0.746	0.830	0.890	0.931
	MLPrior	0.267	0.470	0.649	0.784	0.880	0.942	0.969
	TrajectoryTest	0.285	0.521	0.717	0.863	0.944	0.979	0.992
Geolife	Random	0.099	0.205	0.304	0.405	0.505	0.600	0.702
	DeepGini	0.154	0.294	0.424	0.542	0.652	0.740	0.821
	Entropy	0.152	0.282	0.407	0.522	0.631	0.727	0.811
	PCS	0.154	0.298	0.429	0.543	0.649	0.743	0.826
	VanillaSM	0.159	0.298	0.427	0.545	0.652	0.743	0.824
	MLPrior	0.170	0.317	0.452	0.565	0.668	0.765	0.845
	TrajectoryTest	0.184	0.344	0.490	0.625	0.733	0.831	0.901

Table 5. Average comparison results in terms of NAPFD

Approach	Test Cases Selected						
	10%	20%	30%	40%	50%	60%	70%
Random	0.099	0.177	0.245	0.314	0.369	0.415	0.450
DeepGini	0.361	0.501	0.585	0.644	0.682	0.706	0.722
Entropy	0.347	0.493	0.576	0.635	0.675	0.701	0.716
PCS	0.363	0.503	0.588	0.645	0.683	0.708	0.724
VanillaSM	0.364	0.503	0.587	0.646	0.683	0.708	0.724
MLPrior	0.296	0.461	0.556	0.621	0.667	0.693	0.710
TrajectoryTest	0.413	0.565	0.655	0.719	0.755	0.774	0.785

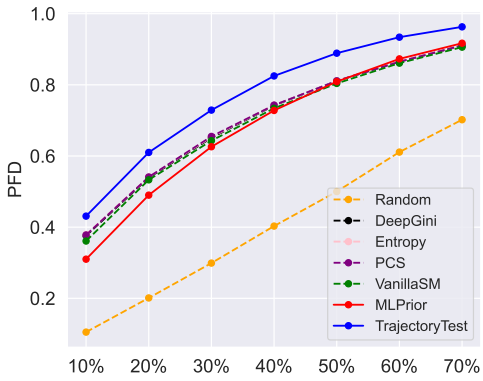


Fig. 4. Average test prioritization effectiveness among TrajectoryTest and the compared test prioritization approaches. X-Axis: the percentage of prioritized tests; Y-Axis: the percentage of detected misclassified tests.

selection, with an average improvement ranging from 7.03% to 9.56%. Below, we present a detailed analysis of the experimental results, structured around the two key findings.

Table 6. Statistical analysis of natural test inputs: p-value (Wilcoxon signed-rank) and effect size (Cliff's delta)

	Random	DeepGini	Entropy	PCS	VanillaSM	MLPrior
TrajectoryTest (p-value)	1.64×10^{-8}	1.26×10^{-07}	3.36×10^{-08}	5.84×10^{-07}	7.09×10^{-07}	2.63×10^{-08}
TrajectoryTest (effect size)	1.0	0.51	0.52	0.49	0.48	0.65

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

Timse cost	**Approach**						
	TrajectoryTest	MLPrior	Random	DeepGini	VanillaSM	PCS	Entropy
Feature generation	5s	3s	0	0	0	0	0
Ranking model training	17s	16s	0	0	0	0	0
Prediction	<1s	<1s	<1s	<1s	<1s	<1s	<1s

Table 8. Feature information of a data point in a trajectory

Person ID	**Latitude**	**Longitude**	**Time**	**Label**
104	39.9661066	116.3406899	2011-03-06 11:19:00	walk

Finding 1: Limitations of MLPrior in the trajectory prediction scenario. We analyze the performance of MLPrior from three evaluation metrics: APFD, PFD, and NAPFD, to assess its effectiveness in the trajectory prediction context.

1) APFD Comparison. Table 3 provides the overall comparison results of test prioritization methods in terms of APFD. In the table, the average effectiveness of each test prioritization approach was calculated as the mean of the scores across the three datasets. The table presents that the state-of-the-art method MLPrior achieves an average APFD of 0.722, which is lower than all of the confidence-based test prioritization methods, whose average APFD values range from 0.732 to 0.739. In the original MLPrior paper [15], MLPrior achieved higher average performance than all the compared confidence-based methods. These experimental results indicate that the state-of-the-art test prioritization method MLPrior is unstable and suboptimal in the context of trajectory prediction (as measured by the APFD metric).

2) PFD Comparison. Figure 3 visually presents the experimental results on two different example subjects from the perspective of the PFD metric. The red solid line represents the state-of-the-art method MLPrior, while the dashed lines correspond to various confidence-based methods and random selection. As shown in the specific case of the Boat dataset with the XGB model (Figure 3(b)), MLPrior consistently performs worse than all confidence-based methods across the entire range. Figure 4 shows the average performance across all 21 subjects, demonstrating that, in most cases, MLPrior either underperforms or performs similarly to all confidence-based methods (e.g., DeepGini, Entropy, PCS, VanillaSM) on average across different test execution percentages. This result further confirms that, despite being considered a state-of-the-art method, MLPrior is less effective than simpler confidence-based strategies in the context of trajectory prediction.

3) NAPFD Comparison. Table 5 presents the average NAPFD scores across different test selection percentages. Under the NAPFD metric, MLPrior also exhibits unstable and suboptimal effectiveness compared to all confidence-based baselines. For instance, at the 50% test execution percentage, MLPrior achieves a NAPFD of 0.667, whereas all confidence-based methods achieve higher scores ranging from 0.675 (Entropy) to 0.683 (PCS/VanillaSM). These results further confirm finding 1.

4) Analysis: The above experiments demonstrate that the state-of-the-art approach MLPrior performs relatively unstable and suboptimal in the context of trajectory prediction compared to its performance in its original paper [15]. In the following, we analyze the reasons behind this instability in its performance.

- **MLPrior's reliance on rich input features.** MLPrior prioritizes test inputs by constructing a composite feature vector for each input, which integrates three different types of information: model mutation features (capturing the impact of model mutations on a test input), input mutation features (capturing the impact of mutations on test inputs), and original attribute features (capturing the spatial relationship between a test input and the decision boundary). These feature vectors serve as the foundation for a learning-to-rank model that estimates the likelihood of misclassification. This approach assumes the availability of meaningful and diverse input attributes to effectively extract informative signals.
- **Trajectory data has limited attribute richness.** In trajectory prediction, the raw input features typically consist only of spatial-temporal coordinates (e.g., latitude, longitude, and timestamps). Table 8 illustrates a sample data point from a trajectory prediction dataset. As shown, the data point with ID 104 contains only the features latitude, longitude, and time. This highlights that the available information for each data point is quite limited.
  As a result, MLPrior's mechanisms, which rely on extracting informative and rich features from relatively high-dimensional feature spaces, become much less effective and stable in this context. In contrast, confidence-based methods such as PCS and Entropy rely solely on output distributions, making them less dependent on the tabular input features, and therefore less affected.
- **MLPrior inherits the limitations of learning-based approaches.** As a learning-based prioritization method, MLPrior requires a sufficient number of misclassified test inputs to construct a balanced and informative training set for the ranking model. However, when the target model achieves high accuracy, the training set for the ranking model can become highly imbalanced. This is because the training labels for the ranking model are determined based on whether a test case is misclassified (labeled as 1) or correctly classified (labeled as 0). If only a small number of test inputs are misclassified, the proportion of positive samples will be extremely low, resulting in a severe class imbalance. Such imbalance can significantly limit the learning effectiveness of the ranking model, reducing the prioritization performance of learning-based methods.

The experimental results also demonstrate that TrajectoryTest outperforms confidence-based test prioritization methods on average. Below, we examine the underlying reasons by analyzing the limitations of confidence-based approaches. We also discuss how TrajectoryTest addresses these limitations.

- **Ineffective under low-accuracy models** Confidence-based methods prioritize test inputs according to the model's predictive uncertainty for each sample. The higher the uncertainty, the more likely the test is considered to be misclassified. However, when the tested model has very low accuracy, this correlation breaks down: high confidence does not necessarily indicate a correct prediction. Even when the model appears "confident", it could be confidently wrong. For example, consider a model with an accuracy of 0.5. For a given input, the model may assign a high confidence score of 0.9, but the prediction is still incorrect. In such cases, confidence-based methods become ineffective.
- **Ignoring informative trajectory signals** Confidence-based methods rely solely on the probabilistic output of the model's final layer, ignoring the rich temporal and structural signals encoded within the trajectory itself. In the context of test input prioritization for trajectory prediction, trajectory data can exhibit these complex patterns that carry valuable information. Neglecting these properties could limit the effectiveness.

To overcome the aforementioned limitations of confidence-based methods, our proposed TrajectoryTest introduces a switching mechanism that avoids using confidence-based strategies when the tested model exhibits low accuracy, and instead adopts a learning-based approach. Moreover, when

operating in the learning-based mode, TrajectoryTest explicitly extracts multiple trajectory-specific features that capture the unique temporal and structural characteristics of trajectories, and leverages these features to perform test input prioritization.

Finding 2: TrajectoryTest outperforms all the existing test prioritization approaches To address the limitations of the state-of-the-art method MLPrior, we propose a novel test prioritization approach tailored for the trajectory prediction scenario, called TrajectoryTest. Below, we present experimental analyses demonstrating that TrajectoryTest outperforms all existing test prioritization methods when applied to trajectory prediction tasks.

1) APFD Comparison Table 2 presents the comparison between TrajectoryTest and existing test prioritization approaches in terms of APFD. The results demonstrate that in most cases, TrajectoryTest achieves the highest effectiveness (with the best-performing methods highlighted in gray). Table 3 further summarizes the overall experimental results. As shown in the table, TrajectoryTest performs best in 16 out of 21 cases. Moreover, the average APFD of TrajectoryTest is 0.791, while the other baseline methods (excluding random selection) have average APFD values ranging from 0.722 to 0.739. Compared to the state-of-the-art method MLPrior, TrajectoryTest achieves an average improvement of 9.56%, with overall improvements ranging from 7.03% to 9.56% across all methods.

2) PFD comparison Table 4, Figure 3, and Figure 4 compare TrajectoryTest with existing approaches from the perspective of the PFD metric. The table shows that TrajectoryTest consistently achieves the highest PFD values across all testing budgets (with the best-performing methods in each case highlighted in gray). Below, we analyze the performance of TrajectoryTest and the existing test prioritization methods from the perspective of each individual dataset.

- **Activity dataset:** TrajectoryTest clearly outperforms all baseline methods, especially under lower testing budgets (e.g., PFD-10 = 0.825), and maintains this lead across all thresholds up to PFD-70 = 0.997. The margins are particularly large compared to the next best methods (e.g., PCS and VanillaSM).
- **Boat dataset:** TrajectoryTest still achieves the highest PFD scores across all testing budgets. Its PFD scores range from 0.285 to 0.992, while the best-performing baseline methods achieve PFD scores ranging from 0.267 to 0.969.
- **Geolife dataset:** TrajectoryTest consistently yields the highest scores from PFD-10 through PFD-70. For instance, at PFD-10 it achieves 0.184 compared to the next best 0.170 (MLPrior), and at PFD-70 it reaches 0.901 compared to the next best 0.845 (MLPrior).

In Figure 3 and Figure 4, the blue solid line represents TrajectoryTest, the red solid line represents the state-of-the-art method MLPrior, and the dashed lines represent the other baseline methods. As shown in the figures, TrajectoryTest consistently outperforms all other methods, maintaining the highest performance across the full range of testing budgets.

3) NAPFD comparision Table 5 presents the average comparison results in terms of NAPFD. As shown in the table, TrajectoryTest consistently achieves the highest NAPFD scores across all test execution budgets. For example, at the 10% budget, TrajectoryTest reaches a NAPFD of 0.413, which is significantly higher than all baselines, including MLPrior (0.296) and the best confidence-based method VanillaSM (0.364). At the 50% budget, TrajectoryTest achieves 0.755, compared to 0.667 for MLPrior and 0.683 for the best-performing confidence-based methods PCS and VanillaSM. This further confirm that TrajectoryTest is a more effective than all the compared baseline methods in the trajectory prediction scenario.

Although our method cannot guarantee the detection of every fault, prior studies have indicated that in practical industrial settings, achieving early detection of the majority of faults is generally considered effective enough [89]. As shown in Table 5, our proposed TrajectoryTest method achieves

a NAPFD score of 0.655 when only the top 30% of test cases are selected, which is significantly higher than other baseline methods (e.g., DeepGini: 0.585; Entropy: 0.576). When 50% of the test cases are selected, the NAPFD score further increases to 0.755, indicating that even under partial testing scenarios, our method can effectively detect the majority of faults at an early stage.

4) Statistical analysis To further validate the statistical significance of the experimental results, we conducted a statistical analysis using the Wilcoxon signed-rank test, along with effect size measurement via Cliff's delta. Table 6 reports the p-values and effect sizes when comparing TrajectoryTest with all baseline methods. The p-values for all comparisons are below 10^{-5} (e.g., 2.63×10^{-8} vs. MLPrior), indicating that the observed effectiveness improvements are statistically significant. Additionally, the effect sizes range from 0.48 to 1.0, all of which exceed the threshold of 0.47, meaning that the improvements achieved by TrajectoryTest represent large practical effects according to widely adopted interpretation guidelines (i.e., Cliff's delta > 0.47 corresponds to a large effect).

5) Efficiency Analysis Table 7 presents a detailed comparison of the time cost among TrajectoryTest and the compared test prioritization approaches. As shown in Table 7, TrajectoryTest requires 5 seconds for feature generation and 17 seconds for ranking model training, in addition to less than 1 second for prediction, resulting in an overall runtime of approximately 23 seconds. The overall runtime of MLPrior is around 20 seconds. In contrast, other approaches, such as Random, DeepGini, VanillaSM, PCS, and Entropy, do not require feature generation or model training, and their prediction time is consistently less than 1 second. Although TrajectoryTest introduces additional time cost compared to confidence-based methods, the extra cost is relatively small (approximately 22 seconds in total), while the resulting improvement in prioritization effectiveness reaches about 7.03% to 9.56% over other approaches. Considering the trade-off between effectiveness and efficiency, TrajectoryTest remains a practical option.

> **Answer to RQ1:** *In the trajectory prediction scenario, the effectiveness of the state-of-the-art method MLPrior is relatively unstable and suboptimal. Our proposed method, TrajectoryTest, significantly outperforms the state-of-the-art method MLPrior and confidence-based approaches, with average improvements ranging from 7.03% to 9.56% over these two categories.*

6.2 RQ2: Comparison of TrajectoryTest and Existing Test Prioritization Techniques on Noisy Test Cases

Objectives: In real-world scenarios, trajectory data can contain noise due to sensor drift, signal interference, or environmental uncertainty [6]. Such noise can significantly impact model behavior and fault detection. To make the empirical evaluation more comprehensive, we not only conducted experiments on natural trajectory datasets (RQ1), but also constructed noisy trajectory datasets to demonstrate the limitations of existing methods in trajectory scenarios and to show that TrajectoryTest consistently outperforms all baselines even under noisy conditions.

Experimental design: We conducted empirical experiments to compare the effectiveness of existing test prioritization methods and TrajectoryTest under noisy trajectory prediction scenarios. In the natural setting (RQ1), we used three trajectory prediction datasets and seven models. For this research question, we constructed noisy datasets based on the natural datasets used in RQ1.

Below, we provide a detailed explanation of the distinction between the natural and noisy scenarios. In particular, we offer an in-depth description of the noisy trajectory prediction scenario.

- **Natural setting** uses trajectory data directly obtained from open-source datasets without any artificial modification or perturbation. These datasets represent clean conditions.

- **Noisy trajectory prediction scenario** is a constructed evaluation context intended to simulate the uncertainty and data corruption. In domains such as autonomous driving, maritime monitoring, or indoor positioning [51, 59], trajectory data collected by sensors can be affected by GPS drift, signal loss, or environmental interference, leading to noisy inputs. To emulate such conditions and systematically evaluate their impact on the effectiveness of test prioritization, we construct this scenario by introducing controlled noise into originally clean datasets (datasets we utilized in RQ1). Specifically, we perturb 50% of the samples using techniques such as random displacement to generate noisy data (see Section 5.3 for detailed descriptions of the noise generation process).

By comparing results under both settings, we aim to thoroughly evaluate the performance of TrajectoryTest, and to understand the limitations of existing test prioritization methods when exposed to noisy data.

Table 9. Effectiveness comparison on noisy datasets in terms of APFD

Data	Model	Approach						
		Random	DeepGini	Entropy	PCS	VanillaSM	MLPrior	**TrajectoryTest**
Activity	DNN	0.489	0.788	0.775	0.797	0.793	0.902	0.921
	DT	0.507	0.726	0.727	0.725	0.728	0.805	0.882
	KNN	0.529	0.890	0.891	0.891	0.892	0.806	0.891
	LR	0.463	0.933	0.931	0.933	0.934	0.825	0.937
	RF	0.538	0.905	0.859	0.929	0.925	0.687	0.938
	XGB	0.464	0.951	0.949	0.950	0.950	0.757	0.961
	LGBM	0.483	0.951	0.953	0.952	0.951	0.701	0.960
Boat	DNN	0.508	0.668	0.666	0.668	0.668	0.766	0.781
	DT	0.502	0.619	0.619	0.619	0.619	0.704	0.739
	KNN	0.503	0.665	0.667	0.667	0.666	0.729	0.757
	LR	0.494	0.641	0.638	0.643	0.642	0.707	0.751
	RF	0.491	0.754	0.745	0.763	0.761	0.733	0.772
	XGB	0.497	0.770	0.769	0.766	0.768	0.708	0.792
	LGBM	0.494	0.769	0.760	0.765	0.768	0.721	0.786
Geolife	DNN	0.503	0.503	0.494	0.521	0.516	0.578	0.593
	DT	0.495	0.572	0.571	0.571	0.572	0.579	0.599
	KNN	0.498	0.535	0.535	0.526	0.532	0.550	0.568
	LR	0.499	0.526	0.481	0.522	0.529	0.576	0.595
	RF	0.494	0.593	0.588	0.619	0.605	0.587	0.612
	XGB	0.503	0.624	0.625	0.615	0.623	0.562	0.606
	LGBM	0.493	0.567	0.568	0.562	0.566	0.585	0.592

Table 10. Overall effectiveness comparison on noisy datasets in terms of APFD

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.497	53.52
DeepGini	0	0.712	7.16
Entropy	1	0.705	8.22
PCS	1	0.714	6.86
VanillaSM	1	0.715	6.71
MLPrior	0	0.694	9.94
TrajectoryTest	18	0.763	-

Results: The experimental results of RQ2 are presented in Table 9, Table 10, Table 11, Table 12 and Table 13. These results report the effectiveness comparison of different test prioritization methods in noisy trajectory prediction contexts. Based on the experimental results, we obtain the following two findings: 1) In the noisy trajectory prediction scenario, MLPrior also performs relatively unstably and suboptimally. 2) In the noisy trajectory prediction scenario, TrajectoryTest also significantly outperforms all existing methods, with an average improvement ranging from 6.71% to 9.94%.

Table 11. Effectiveness comparison on noisy datasets in terms of PFD

Data	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
Activity	Random	0.090	0.174	0.281	0.382	0.499	0.598	0.687
	DeepGini	0.655	0.848	0.902	0.939	0.941	0.954	0.967
	Entropy	0.623	0.806	0.887	0.934	0.941	0.954	0.968
	PCS	0.692	0.845	0.906	0.935	0.940	0.954	0.969
	VanillaSM	0.686	0.848	0.902	0.935	0.942	0.955	0.967
	MLPrior	0.474	0.649	0.746	0.818	0.877	0.922	0.946
	TrajectoryTest	0.790	0.939	0.975	0.987	0.991	0.992	0.996
Boat	Random	0.096	0.196	0.299	0.403	0.500	0.596	0.690
	DeepGini	0.236	0.429	0.597	0.724	0.815	0.882	0.925
	Entropy	0.238	0.424	0.587	0.720	0.816	0.883	0.925
	PCS	0.223	0.430	0.604	0.723	0.819	0.882	0.925
	VanillaSM	0.230	0.433	0.601	0.724	0.815	0.882	0.925
	MLPrior	0.245	0.451	0.620	0.757	0.856	0.924	0.963
	TrajectoryTest	0.262	0.495	0.675	0.820	0.920	0.971	0.989
Geolife	Random	0.099	0.201	0.303	0.403	0.502	0.598	0.698
	DeepGini	0.128	0.248	0.366	0.474	0.583	0.682	0.781
	Entropy	0.126	0.244	0.357	0.465	0.572	0.676	0.773
	PCS	0.127	0.257	0.373	0.482	0.587	0.689	0.782
	VanillaSM	0.129	0.254	0.375	0.479	0.587	0.686	0.784
	MLPrior	0.133	0.263	0.388	0.502	0.610	0.709	0.798
	TrajectoryTest	0.140	0.276	0.403	0.523	0.639	0.745	0.838

Table 12. Average comparison results on noisy datasets in terms of NAPFD

Approach	Test Cases Selected						
	10%	20%	30%	40%	50%	60%	70%
Random	0.094	0.181	0.259	0.318	0.373	0.420	0.455
DeepGini	0.334	0.473	0.558	0.617	0.656	0.683	0.701
Entropy	0.324	0.460	0.548	0.608	0.650	0.676	0.695
PCS	0.342	0.477	0.562	0.620	0.660	0.686	0.704
VanillaSM	0.342	0.480	0.563	0.620	0.659	0.685	0.704
MLPrior	0.273	0.412	0.510	0.579	0.629	0.661	0.680
TrajectoryTest	0.384	0.530	0.617	0.676	0.716	0.740	0.753

Table 13. Statistical analysis of noisy test inputs: p-value (Wilcoxon signed-rank) and effect size (Cliff's delta)

	Random	DeepGini	Entropy	PCS	VanillaSM	MLPrior
TrajectoryTest (p-value)	1.61×10^{-8}	9.56×10^{-08}	1.09×10^{-07}	2.81×10^{-07}	1.77×10^{-07}	2.61×10^{-08}
TrajectoryTest (effect size)	1.0	0.49	0.51	0.45	0.46	0.62

Finding 1: MLPrior still shows limitations in the noisy trajectory prediction scenario.

1) APFD comparison Table 10 provides the overall comparison results of test prioritization methods on the noisy datasets in terms of APFD. The table shows that the state-of-the-art method MLPrior achieves an average APFD of 0.694, which is lower than all of the confidence-based test prioritization methods, whose average APFD values range from 0.705 to 0.715. These results demonstrate that, under the noisy trajectory prediction scenario, the effectiveness of MLPrior remains unstable and suboptimal.

2) PFD comparison Table 11 further compares the effectiveness of test prioritization methods on noisy datasets in terms of the PFD metric. On the Activity dataset, MLPrior underperforms in all cases compared to the confidence-based baselines. Specifically, MLPrior's PFD ranges from 0.474 to 0.946, which is consistently lower than the corresponding values of all confidence-based methods: DeepGini (0.655–0.967), Entropy (0.623–0.968), PCS (0.692–0.969), and VanillaSM (0.686–0.967). This reflects the instability of the MLPrior method.

3) NAPFD comparison Table 12 presents the average NAPFD values of all test prioritization methods across different test execution percentages on noisy datasets. As shown in the table, MLPrior consistently underperforms compared to all confidence-based baselines. Specifically, MLPrior's NAPFD values range from 0.273 to 0.680, whereas the confidence-based methods achieve higher and more stable performance across all budgets: DeepGini (0.334–0.701), Entropy (0.324–0.695), PCS (0.342–0.704), and VanillaSM (0.342–0.704). This further confirms the relative instability and limited effectiveness of MLPrior in noisy trajectory prediction scenarios.

Finding 2: TrajectoryTest outperforms all existing test prioritization methods in the noisy trajectory prediction scenario.

1) APFD comparison Table 9 presents the comparison between TrajectoryTest and existing test prioritization approaches in terms of APFD on noisy datasets. The results demonstrate that in most cases (18 out of 21), TrajectoryTest achieves the highest effectiveness. Table 10 further summarizes the overall experimental results. As shown in the table, TrajectoryTest outperforms all baseline methods. The average APFD of TrajectoryTest reaches 0.763, while the other baselines (excluding random selection) have average APFD values ranging from 0.694 to 0.715. Compared to the state-of-the-art method MLPrior, TrajectoryTest achieves an average improvement of 9.94%. The overall improvements of TrajectoryTest over all test prioritization approaches range from 6.71% to 9.94%. These results strongly indicate that TrajectoryTest outperforms all existing test prioritization approaches.

2) PFD comparison Table 11 shows the effectiveness comparison on noisy datasets in terms of PFD. As shown in the table, TrajectoryTest consistently achieves the highest PFD scores across all thresholds (PFD-10 to PFD-70) and datasets (Activity, Boat, and Geolife). These results further indicate that TrajectoryTest maintains consistently strong prioritization performance compared to existing test prioritization approaches.

These results suggest that TrajectoryTest is robust to noise in test data, which is common in real-world trajectory prediction scenarios due to imperfect sensors, incomplete data, or mislabeled outcomes. Notably, this robustness is consistently observed across all three datasets (Activity, Boat, and Geolife). Across all noisy datasets, TrajectoryTest consistently outperforms all baseline methods. This highlights its ability to generalize well to real-world and imperfect data and to reliably prioritize test inputs in the presence of noise.

3) NAPFD comparison Table 12 presents the average comparison results on noisy datasets in terms of NAPFD. As shown in the table, TrajectoryTest consistently achieves the highest NAPFD scores across all test execution ratios, outperforming all baseline methods. Specifically, the NAPFD scores of TrajectoryTest range from 0.384 to 0.753. In comparison, the best-performing baseline method, PCS, achieves scores ranging from 0.342 to 0.704.

4) Statistical analysis To validate the statistical significance of the experimental results on noisy datasets, we also conducted a statistical analysis. The results are presented in Table 13. The p-values for all comparisons are below 10^{-5} (e.g., 2.61×10^{-8} vs. MLPrior). This indicates that TrajectoryTest statistically and significantly outperforms all baselines on noisy datasets. Additionally, the effect sizes range from 0.45 to 1.0. According to the standard interpretation thresholds, an effect size of $|\delta| < 0.15$ is considered negligible, $0.15 \leq |\delta| < 0.33$ is small, $0.33 \leq |\delta| < 0.47$ is medium, and $|\delta| \geq 0.47$ is large. This indicates that all performance improvements of TrajectoryTest over the baselines are non-negligible, falling within the medium to large effect size range.

> **Answer to RQ2:** *In the noisy trajectory prediction scenario, MLPrior also performs relatively unstably and suboptimally. TrajectoryTest still significantly outperforms MLPrior and confidence-based approaches, achieving average improvements ranging from 6.71% to 9.94% over these methods.*

6.3 RQ3: Ablation Study

Objectives: We aim to understand the contribution of each individual component within TrajectoryTest to its overall effectiveness. TrajectoryTest consists of multiple integrated components. This research question investigates whether each component contributes to the effectiveness of TrajectoryTest through an ablation study.

Experimental design: The original TrajectoryTest framework contains two major components: trajectory temporal features (TTF) and MLPrior-derived features (MLF). To assess the individual contributions of these components to the overall effectiveness of TrajectoryTest, we conducted a carefully designed ablation study, following the methodology used in prior work [20].

More specifically, we removed one component at a time and evaluated the effectiveness of TrajectoryTest under these modified configurations. For example, to assess the contribution of trajectory temporal features, we excluded TTF while keeping all other processes unchanged and compared the effectiveness of TrajectoryTest before and after the removal. Each ablated variant was evaluated using the same seven models and three trajectory prediction datasets as used in previous RQs.

Table 14. Ablation study results

Approach	Dataset			Average
	Activity	Boat	Geolife	
TrajectoryTest w/o TTF	0.929	0.762	0.621	0.771
TrajectoryTest w/o MLF	0.893	0.774	0.634	0.767
TrajectoryTest	0.937	0.782	0.655	0.791

Results: The results of our ablation study are presented in Table 14. In this table, "w/o" stands for "without." For example, "TrajectoryTest w/o TTF" refers to the variant of our method without the use of trajectory temporal features (TTF), while "TrajectoryTest w/o MLF" refers to the version excluding MLPrior-derived features (MLF). Table 14 presents that the complete TrajectoryTest model achieves the highest average effectiveness across all three datasets (Activity, Boat, and Geolife), reaching an average APFD of 0.791. Removing either TTF or MLF leads to noticeable performance degradation, demonstrating that both components play important roles. Specifically, removing TTF results in a performance drop to 0.771, while removing MLF reduces the average to 0.767. These findings suggest that the both two types of features contribute to the effectiveness of TrajectoryTest.

> **Answer to RQ3:** *The ablation study demonstrates that each component contributes to the effectiveness of TrajectoryTest.*

6.4 RQ4: Contribution of the Adaptive Switching Mechanism to the Effectiveness of TrajectoryTest

Objectives: TrajectoryTest employs an adaptive switching mechanism that switches between confidence-based and learning-based prioritization methods based on the error ratio (which measures the proportion of misclassified samples by the target model within the training dataset), with the goal of ensuring more stable and effective performance. The details of this mechanism can be found in Section 4. This research question aims to quantify the impact of this mechanism. Specifically, we implement a non-adaptive variant of TrajectoryTest that always applies the learning-based ranking branch, and compare it with the full adaptive version. By evaluating both versions, we aim to analyze the specific contribution of the adaptive switching strategy to overall performance.

Experimental design: To evaluate the contribution of the adaptive switching mechanism in TrajectoryTest, we designed the following experiments. We first implemented a non-adaptive variant

of TrajectoryTest, in which the error ratio is not computed, and the learning-based prioritization method based on trajectory information is always applied. In contrast, the adaptive version (i.e., the default setting) dynamically switches between confidence-based and learning-based ranking strategies depending on the model's error ratio. We compare the two variants across multiple models and datasets. By analyzing their performance differences, we aim to assess the practical benefit of the adaptive switching strategy.

Table 15. Effectiveness comparison between adaptive and non-adaptive variants of TrajectoryTest

Approach	Dataset			Average
	Activity	Boat	Geolife	
TrajectoryTest (non-adaptive variant)	0.903	0.775	0.638	0.772
TrajectoryTest	0.937	0.782	0.655	0.791

Results: The results of RQ4 are presented in Table 15, which shows the effectiveness comparison between the original TrajectoryTest and non-adaptive variants across three benchmark datasets. The cells highlighted in gray indicate the best-performing version of TrajectoryTest for each case. The results show that the adaptive version (default) consistently outperforms the non-adaptive variant across all three datasets. For example, on the Activity dataset, the adaptive version achieves an effectiveness score of 0.937, compared to 0.903 for the non-adaptive variant, resulting in an absolute gain of 0.034. Similarly, on the Geolife dataset, the adaptive version scores 0.655, outperforming the non-adaptive variant's 0.638, with an average improvement of 0.017. From the perspective of the average across all cases, the original TrajectoryTest with the adaptive switching mechanism achieves an APFD of 0.791, exceeding the non-adaptive variant's 0.772 by 0.019.

> **Answer to RQ4:** *The original TrajectoryTest with the adaptive switching mechanism consistently performs better than the non-adaptive variant across all the datasets, indicating that the adaptive mechanism contributes to the effectiveness of TrajectoryTest.*

6.5 RQ5: Impact of the Error Ratio Threshold on the Effectiveness of TrajectoryTest

Objectives: The adaptive switching mechanism in TrajectoryTest relies on an error ratio threshold to determine whether to use confidence-based or trajectory-informed learning-based prioritization. This threshold controls when the system switches strategies, and its selection could influence performance. This research question investigates how sensitive TrajectoryTest is to different threshold values. Through this sensitivity analysis, we aim to evaluate the robustness of TrajectoryTest and provide practical guidance for selecting an appropriate threshold for trajectory prediction scenarios.
Experimental design: To assess the sensitivity of TrajectoryTest to the error ratio threshold used in the adaptive switching mechanism, we conduct a controlled experiment by varying the threshold value while keeping all other components fixed. Specifically, we evaluate six different threshold values: 0.01, 0.03, 0.05, 0.10, 0.20, and 0.30. We report the effectiveness of TrajectoryTest under each setting across three benchmark datasets used in the previous research questions: Activity, Boat, and Geolife. By analyzing TrajectoryTest's effectiveness at different threshold levels, we aim to examine its robustness and stability.
Results: Table 16 presents the effectiveness of TrajectoryTest under six different error ratio thresholds. The cells highlighted in gray indicate the best-performing setting for each dataset and the overall average. The results show that TrajectoryTest is relatively robust when the error ratio threshold is set between 0.01 and 0.10. Within this range, the effectiveness remains stable with only minor variations. The best performance is achieved at a threshold of 0.03, with an average APFD of 0.792. When the threshold is set to 0.10, the APFD drops slightly to 0.784, the lowest within this

Table 16. Effectiveness of TrajectoryTest under different error ratio thresholds

Error Ratio	Dataset			Average
	Activity	Boat	Geolife	
0.01	0.926	0.784	0.657	0.789
0.03	0.938	0.782	0.656	0.792
0.05	0.937	0.782	0.655	0.791
0.10	0.914	0.781	0.656	0.784
0.20	0.896	0.762	0.647	0.768
0.30	0.896	0.707	0.646	0.749

range. The variation is within 0.008, indicating that TrajectoryTest is not highly sensitive to the specific threshold value when it falls in this lower range.

However, when the error ratio threshold increases to 0.20 and 0.30, the effectiveness drops more noticeably. Specifically, the APFD decreases to 0.768 and 0.749, respectively, showing reductions of 0.024 and 0.043 compared to the peak value. This is because setting the error ratio threshold too high (e.g., 0.20 or 0.30) could cause cases where learning-based ranking is actually appropriate (i.e., the training data for the ranking model is not imbalanced) to be mistakenly switched to confidence-based methods. In our framework, the error ratio is used to determine whether to apply the learning-based ranking strategy. The purpose of this mechanism is to ensure that when the sample distribution is highly imbalanced, specifically, when the number of misclassified examples is too small (i.e., a very low error ratio), the system switches to the confidence-based method. However, if the threshold is set too high, then even balanced datasets may incorrectly trigger the confidence-based method. As a result, the overall effectiveness of the prioritization is negatively affected.

> **Answer to RQ5:** *TrajectoryTest is relatively robust to the error ratio threshold when it is set between 0.01 and 0.10, but overly high thresholds (e.g., 0.20 or 0.30) could lead to drops in its prioritization effectiveness.*

7 DISCUSSION

In this section, we provide a discussion of TrajectoryTest from two crucial perspectives: 1) the generality of the proposed framework, which highlights its applicability beyond the evaluated settings, and 2) the potential threats to validity, which critically reflect on the limitations of our study and outline the corresponding mitigation strategies we adopted.

7.1 Generality of TrajectoryTest

In this paper, we introduce TrajectoryTest, a trajectory-specific test input prioritization framework that integrates trajectory-aware information with MLPrior-derived features, and adaptively switches between learning-based and confidence-based strategies. Our evaluation across seven models and three representative trajectory datasets demonstrates that TrajectoryTest consistently improves the effectiveness of test input prioritization.

Although TrajectoryTest is evaluated on specific datasets and models, its underlying design principles are more generalizable and can be adapted to other trajectory prediction models and datasets, as long as the data is in a tabular format. This generality highlights that the framework is not confined to a single task or benchmark, but instead provides a flexible foundation for test prioritization across a broader range of trajectory prediction problems. In addition, the integration of trajectory-aware features with MLPrior-derived representations establishes a modular design. This modularity ensures that the framework is not static: new features can be incorporated without

disrupting the overall structure. As a result, TrajectoryTest can continually evolve and remain effective as new modeling paradigms and data types emerge.

Below, we provide a detailed explanation of why our method remains effective across a broader range of trajectory prediction datasets, linking its performance gains to specific trajectory characteristics. Furthermore, we identify cases where confidence-based methods are still sufficient.

7.1.1 Trajectory-Aware Features and Performance Gains.

TrajectoryTest is designed to leverage trajectory-aware information to guide test prioritization. As shown in Table 2, compared to MLPrior (which also trains a ranking model but does not incorporate any trajectory features), TrajectoryTest consistently achieves better performance across different datasets and models. For example, on the Activity dataset, TrajectoryTest improves over MLPrior in nearly all models, such as increasing APFD from 0.861 to 0.905 for the Tree model and from 0.712 to 0.948 for the RF model. Similar trends appear in the Boat dataset, where TrajectoryTest raises the APFD of the LGBM model from 0.734 to 0.799, and improves the RF model from 0.766 to 0.799. On the GeoLife dataset, TrajectoryTest also outperforms MLPrior, for example, improving LGBM from 0.612 to 0.669 and XGB from 0.601 to 0.675. These improvements highlight the contribution of trajectory-specific spatiotemporal patterns, which provide richer behavioral information. As a result, TrajectoryTest is able to more accurately identify potentially misclassified samples in trajectory prediction scenarios, leading to more effective prioritization.

7.1.2 Confidence-Based Methods in High-Accuracy Settings.

When the model achieves very high accuracy on a trajectory prediction dataset, the training data for TrajectoryTest's ranking model can become highly imbalanced (i.e., too few test samples are labeled as misclassified), which negatively affects its effectiveness. In such cases, confidence-based methods remain sufficient for effective test input prioritization. As shown in Table 2, on the Activity dataset, confidence-based methods such as PCS and VanillaSM deliver highly competitive results across multiple settings. For example, on the XGB model, PCS achieves an APFD of 0.970, nearly matching TrajectoryTest(0.969). On the RF model, PCS reaches 0.947, while VanillaSM and Entropy also exceed 0.9, which is comparable to the performance of TrajectoryTest(0.948). For LR and KNN, confidence-based methods achieve APFD scores as high as 0.946, further demonstrating their effectiveness for test prioritization in high-accuracy model scenarios.

7.2 Threats to Validity

Threats to Internal Validity. The internal threats to validity mainly lie in the implementation of existing test prioritization framework and the baseline methods (e.g., MLPrior, DeepGini, Entropy and random selection). To mitigate this threat, we carefully referred to the official code repositories and experimental settings reported in the original papers. For reproducibility and consistency, we used mainstream libraries such as PyTorch and scikit-learn (cf. Section 5.5 for details), and ensured consistent data preprocessing and model training procedures. Additionally, the randomness inherent in model training could affect the results. To address this, we repeated all experiments ten times and reported the average results. Moreover, we conducted statistical analysis and calculated the p-value and effect size to demonstrate whether our proposed method outperforms the existing test prioritization methods.

Threats to External Validity. External threats to validity in our study primarily arise from the datasets and models chosen for evaluation. Currently, we conducted experiments on three trajectory prediction datasets. To mitigate this threat, our datasets cover different domains, including transportation, maritime activities, and human behavior recognition, and we used diverse classification models. The adopted datasets exhibit varying temporal and spatial characteristics to enhance the representativeness of our evaluation.

Threats to Construct Validity. Construct validity threats mainly lie in the measurements we consider in evaluating the effectiveness of test input prioritization. Specifically, we assess the correlation between the studied metrics and the actual performance of prioritization techniques. To mitigate this threat, we adopted three widely used and well-established metrics: Average Percentage of Fault Detection (APFD), Percentage of Faults Detected (PFD), and Normalized Average Percentage of Fault Detection (NAPFD). These metrics are commonly employed in software testing literature and provide a quantitative basis for comparing the effectiveness of different prioritization approaches.

Threats to Conclusion Validity. Conclusion validity threats mainly lie in the risk that observed performance differences across methods may be due to random variation rather than actual improvements from the proposed technique. To mitigate this threat, we performed all experiments ten times and reported the average results to reduce the impact of randomness in model training and evaluation. Additionally, we applied rigorous statistical analysis, including the Wilcoxon signed-rank test and Cliff's delta, to ensure that the performance improvements of TrajectoryTest over baseline methods are statistically significant and not due to chance. These efforts strengthen the credibility of our conclusions regarding the effectiveness of the proposed approach. Furthermore, TrajectoryTest relies on a fixed threshold (error ratio = 0.05) to determine the switching strategy. To reduce the risk that our conclusions may depend on this specific threshold, we conducted additional experiments (cf. Section 6.5) to assess the sensitivity of our results under different threshold values. These analyses support the robustness of our findings.

8 RELATED WORK

In this section, we first review recent progress in deep neural network testing, and then introduce the research on test input prioritization in traditional software.

8.1 Deep Neural Network Testing

Beyond test prioritization, recent research in deep neural network testing has explored several critical directions, including accuracy estimation and testing adequacy measurement, both of which aim to improve testing efficiency [9, 24, 27, 34, 36, 41, 53, 57, 63].

Accuracy estimation aims to estimate the overall performance of a DNN using only a subset of the test data, thereby reducing the labeling cost. Li *et al.* [53] proposed Cross Entropy-based Sampling (CES), which selects representative samples by minimizing the cross-entropy in the model's latent space. Chen *et al.* [9] introduced PACE, a clustering-based method that leverages the MMD-critic algorithm to choose prototypes from each cluster. Later works such as DeepReduce [100] and DeepSample [27] adopted coverage-aware and stratified sampling strategies to further enhance representativeness and cost-efficiency. More recently, labeling-free approaches have emerged. Aries [34] estimates accuracy by analyzing confidence-based distributions without requiring any labels of the tests. Its core principle is that samples closer to the decision boundary tend to have less reliable predictions. Specifically, Aries leverages uncertainty estimated via dropout inference to partition the data into confidence-based regions, and infers the overall performance on unlabeled data by referencing the accuracy of corresponding regions in labeled data.

Test adequacy measurement, originally designed for test input generation and evaluation, has also been widely used to identify potentially misclassified inputs. Pei *et al.* [63] first introduced the neuron coverage metric, which measures the proportion of activated neurons during DNN execution and leverages this information to guide test input generation via differential testing among multiple DNNs. Ma *et al.* [57] extended this concept through DeepGauge, proposing a suite of fine-grained adequacy criteria at both the neuron and layer levels, such as k-multisection neuron coverage and top-k neuron patterns. Recent studies have further enriched the coverage landscape by introducing

criteria based on neuron importance [26], neuron path abstraction [86], and probabilistic state transitions between neurons [70], all of which aim to improve the interpretability of test adequacy and strengthen fault detection capabilities in deep learning systems.

8.2 Test Input Prioritization for Traditional Software

Test prioritization was originally introduced in the context of traditional software testing, aiming to determine the optimal order for executing test cases to enable earlier fault detection [4, 18, 31, 32, 38, 52, 62, 69, 69, 72, 79, 95].

Shin *et al.* [72] proposed a diversity-aware mutation adequacy criterion to guide test case prioritization and empirically evaluated mutation-based prioritization techniques using large-scale developer-written test cases. Jahangirova *et al.* [38] conducted an empirical study evaluating how mutation-based test prioritization techniques perform across a diverse set of projects and mutation operators.

Sharif *et al.* [69] introduced DeepOrder, a regression-based deep learning model designed for test case prioritization in continuous integration (CI) environments. The model learns from historical test execution data to assign priorities to test cases using a multi-layer neural network. DeepOrder improves over previous methods by enabling the use of long test histories (beyond just a few cycles), and its empirical evaluation on industrial datasets demonstrates superior fault detection effectiveness.

Pan *et al.* [62] conducted literature review on ML-based test case selection and prioritization for traditional software testing. Analyzing 29 primary studies, their review categorizes existing work into supervised learning, unsupervised learning, reinforcement learning, and NLP-based approaches. The study highlights that while supervised ranking models (e.g., MART, SVM-Rank) remain among the most accurate techniques, many of them rely on batch learning and lack incremental learning capabilities, which limits their adaptability in Continuous Integration (CI) environments.

Bagherzadeh *et al.* [4] investigated the use of reinforcement learning (RL) for test case prioritization in continuous integration settings. They modeled TCP as a sequential decision-making problem and explored three ranking formulations: pointwise, pairwise, and listwise. Their approach continuously learns an adaptive prioritization policy by interacting with a simulated CI environment using execution history and lightweight code features.

Zhang *et al.* [95] proposed OCP, a novel test case prioritization technique that incorporates a partial attention mechanism into the additional-greedy strategy. Rather than considering all test cases in each iteration, OCP maintains and reuses priority scores from previous iterations to focus computation on a subset of promising candidates.

8.3 Empirical study for DNN Test Optimization

In recent years, a growing number of empirical studies [7, 8, 14, 33, 37, 71, 76, 97] have explored ways to optimize DNN testing, focusing on improving fault detection, reducing labeling costs, and enhancing robustness. These studies typically evaluate or propose test prioritization, test case selection, or metric effectiveness.

Dong *et al.* [19] conducted one of the earliest large-scale empirical studies to assess the correlation between coverage metrics and DNN robustness. Their findings indicated limited correlation between neuron coverage and robustness, suggesting that coverage-based testing may not reliably reflect model quality.

Feng *et al.* [24] proposed DeepGini and conducted an empirical evaluation demonstrating that DeepGini is significantly more efficient and effective than traditional coverage-based methods in detecting misclassified inputs.

Hu *et al.* [33] conducted a comprehensive empirical study to investigate the effectiveness of test selection techniques for enhancing deep learning systems under distribution shifts. Their findings reveal that 1) retraining with both the original training data and selected new data achieves superior performance compared to using only the newly selected data, and 2) existing selection metrics are sensitive to data distribution changes and class bias. To address these issues, they proposed DAT, a novel distribution-aware test selection metric that leverages out-of-distribution detection to balance in-distribution uncertainty with out-of-distribution representativeness.

Chen *et al.* [10] conducted an empirical study on performance testing and introduced DeepPerform, which generates test inputs that reveal input-dependent performance bottlenecks in adaptive neural networks. Their empirical evaluation demonstrated notable increases in computation and energy consumption, highlighting a crucial testing dimension.

Demir *et al.* [16] conducted an empirical study on test selection for deep neural networks and proposed a meta-model-based approach that integrates multiple uncertainty metrics. Their evaluation on in-distribution and out-of-distribution test datasets demonstrates that the proposed method effectively prioritizes fault-revealing inputs by leveraging both model confidence and ensemble disagreement, outperforming state-of-the-art techniques across diverse test scenarios.

8.4 Trajectory Prediction in Intelligent Systems

Trajectory prediction is a fundamental capability in many intelligent systems deployed in safety-critical domains such as autonomous driving, maritime monitoring, and human activity recognition [1, 17, 68, 96]. Prior work has proposed a variety of models for trajectory prediction, ranging from classical methods such as Kalman filters and Bayesian networks to deep learning approaches like recurrent neural networks (RNNs), convolutional architectures, and attention-based models [1, 68]. In maritime scenarios, trajectory prediction is utilized for monitoring vessel behavior, detecting illegal fishing or piracy, and ensuring navigational safety. AIS-based modeling approaches [46] are now routinely used by maritime authorities to predict vessel routes and identify anomalies. Li *et al.* [45] conducted a comprehensive analysis of ship trajectory prediction methods using AIS data, demonstrating how predictive models can effectively aid in identifying abnormal ship behaviors and reducing maritime risks such as collision and stranding. Evmides *et al.* [21] developed a deep learning framework that evaluates multiple recurrent neural architectures for vessel movement forecasting, providing insights into temporal dynamics and model performance across varying prediction horizons. Shin *et al.* [73] proposed the AIS-ACNet, a convolutional deep learning model that incorporates vessel dynamics and auxiliary prediction tasks, achieving improved accuracy for trajectory forecasting in real-world AIS datasets.

In aviation, air traffic management systems rely on trajectory prediction to ensure safe separation and optimize flight scheduling [92]. For example, Kuang *et al.* [43] proposed a spatiotemporal graph convolutional network that models aircraft interactions for four-dimensional trajectory prediction, improving prediction accuracy in dense terminal areas. Pepper *et al.* [64] developed fast surrogate models to adaptively predict aircraft trajectories in en-route airspace, which demonstrates improved climb/descent phase estimation and shows potential for use in conflict detection. Tang *et al.* [77] investigated the role of high-precision 4D trajectory data and trajectory-based operations (TBO) for optimizing airspace utilization, highlighting how accurate prediction supports collaborative decision making across ATM stakeholders.

In sports analytics and healthcare, modeling human motion trajectories is used to assess performance or detect abnormal movement patterns [12, 88]. For example, in healthcare, gait trajectory modeling using wearable sensors allows for the early detection of neurological or musculoskeletal disorders, such as Parkinson's disease or post-stroke abnormalities, by identifying deviations from typical walking patterns [12]. These diverse applications highlight that trajectory prediction is a

foundational capability for intelligent systems operating in dynamic and interactive environments. As such, it continues to receive significant research attention across disciplines, including computer vision, robotics, transportation, and geospatial intelligence [35, 40, 74].

9 CONCLUSION

In this paper, we investigate the challenge of high labeling costs in the trajectory prediction domain. Through a detailed analysis of the state-of-the-art test input prioritization method, MLPrior, we identify two key limitations in the context of trajectory prediction: 1) the limited richness of input features in trajectory data restricts the effectiveness of MLPrior's feature-based ranking mechanism; and 2) MLPrior inherits the limitations of learning-based approaches. Specifically, when the training dataset used for the ranking model is imbalanced (i.e., when the number of misclassified tests is very small), its effectiveness is significantly reduced. To this end, we propose TrajectoryTest, a novel test input prioritization approach that incorporates trajectory-specific information and adopts an adaptive strategy to improve prioritization effectiveness. Specifically, TrajectoryTest integrates two core design principles: **1) Adaptive strategy based on error ratio.** TrajectoryTest adaptively switches between uncertainty-based prioritization and learning-to-rank, depending on the model's error ratio, thus avoiding the effect of severe class imbalance; **2) Trajectory-aware prioritization.** TrajectoryTest enhances test prioritization by incorporating enriched trajectory features (e.g., dwell time, visit count, heading, speed), and combining features from the state-of-the-art method MLPrior to enable effective prioritization for trajectory prediction context. Based on the results of the empirical evaluation, we first confirm the limitations of MLPrior, finding that its effectiveness in the trajectory prediction scenario is relatively unstable and suboptimal. We then demonstrate that TrajectoryTest consistently outperforms MLPrior (the state-of-the-art method) and confidence-based approaches, achieving average improvements ranging from 7.03% to 9.56% on natural datasets and from 6.71% to 9.94% on noisy datasets.

Availability. Our replication package is available at https://zenodo.org/records/17035802

ACKNOWLEDGEMENTS

This work was supported by the Luxembourg National Research Fund (FNR) through the CORE project C23/IS/18182513/MiCE, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 949014), and Fundamental Research Funds for the Central Universities (AE89991/478).

REFERENCES

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. 2016. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 961–971.

[2] Alibaba Cloud Tianchi Lab. 2023. BoAT: Behavior of AIS-equipped Trawlers. https://tianchi.aliyun.com/competition/entrance/231768/information. Accessed: 2025-07-31.

[3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3–4.

[4] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand. 2021. Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering* 48, 8 (2021), 2836–2856.

[5] Samuele Capobianco, Leonardo M Millefiori, Nicola Forti, Paolo Braca, and Peter Willett. 2021. Deep learning methods for vessel trajectory prediction based on recurrent neural networks. *IEEE Trans. Aerospace Electron. Systems* 57, 6 (2021), 4329–4346.

[6] Rohan Chandra, Uttaran Bhattacharya, Christian Roncal, Aniket Bera, and Dinesh Manocha. 2019. Robusttp: End-to-end trajectory prediction for heterogeneous road-agents in dense traffic with noisy sensor inputs. In *Proceedings of the 3rd ACM Computer Science in Cars Symposium*. 1–9.

[7] Jinyin Chen, Jie Ge, and Haibin Zheng. 2023. ActGraph: prioritization of test cases based on deep neural network activation graph. *Automated Software Engineering* 30, 2 (2023), 28.

[8] Jialuo Chen, Jingyi Wang, Xiyue Zhang, Youcheng Sun, Marta Kwiatkowska, Jiming Chen, and Peng Cheng. 2024. 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*. 895–906.

[9] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical accuracy estimation for efficient deep neural network testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 4 (2020), 1–35. https://doi.org/10.1145/3394112

[10] Simin Chen, Mirazul Haque, Cong Liu, and Wei Yang. 2022. Deepperform: An efficient approach for performance testing of resource-constrained neural networks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

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

[12] Courtney Chheng and Denise Wilson. 2021. Abnormal gait detection using wearable hall-effect sensors. *Sensors* 21, 4 (2021), 1206.

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

[14] Xueqi Dang, Yinghua Li, Mike Papadakis, Jacques Klein, Tegawendé F Bissyandé, and Yves Le Traon. 2023. Graphprior: mutation-based test input prioritization for graph neural networks. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–40.

[15] Xueqi Dang, Yinghua Li, Mike Papadakis, Jacques Klein, Tegawendé F Bissyandé, and Yves Le Traon. 2024. Test input prioritization for machine learning classifiers. *IEEE Transactions on Software Engineering* 50, 3 (2024), 413–442.

[16] Demet Demir, Aysu Betin Can, and Elif Surer. 2024. Test selection for deep neural networks using meta-models with uncertainty metrics. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 678–690.

[17] Nachiket Deo and Mohan M Trivedi. 2018. Convolutional social pooling for vehicle trajectory prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 1468–1476.

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

[19] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jinsong Dong, and Ting Dai. 2020. An empirical study on correlation between coverage and robustness for deep neural networks. In *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 73–82.

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

[21] Nicos Evmides, Michalis P Michaelides, and Herodotos Herodotou. 2025. Vessel Trajectory Prediction with Deep Learning: Temporal Modeling and Operational Implications. *Journal of Marine Science and Engineering* 13, 8 (2025), 1439.

[22] Junliang Fan, Xin Ma, Lifeng Wu, Fucang Zhang, Xiang Yu, and Wenzhi Zeng. 2019. Light Gradient Boosting Machine: An efficient soft computing model for estimating daily reference evapotranspiration with local and external meteorological data. *Agricultural water management* 225 (2019), 105758.

[23] Zilin Fang, David Hsu, and Gim Hee Lee. 2025. Neuralized Markov Random Field for Interaction-Aware Stochastic Human Trajectory Prediction. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=r3cEOVj7Ze

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

[25] Daniel Garcia-Gonzalez, Daniel Rivero, Enrique Fernandez-Blanco, and Miguel R Luaces. 2023. Deep learning models for real-life human activity recognition from smartphone sensor data. *Internet of Things* 24 (2023), 100925.

[26] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-driven deep learning system testing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 702–713. https://doi.org/10.1145/3377811.3380391

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

[28] Dongyue Guo, Zheng Zhang, Bo Yang, Jianwei Zhang, Hongyu Yang, and Yi Lin. 2024. Integrating spoken instructions into flight trajectory prediction to optimize automation in air traffic control. *Nature Communications* 15, 1 (2024), 9662.

[29] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. 2018. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2255–2264.

[30] Seyed Mohammad Hashemi, Ruxandra Mihaela Botez, and Georges Ghazi. 2024. Robust trajectory prediction using random forest methodology application to UAS-S4 ehécatl. *Aerospace* 11, 1 (2024), 49.

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

[32] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International conference on software engineering*. 523–534.

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

[34] Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Mike Papadakis, Lei Ma, and Yves Le Traon. 2023. Aries: Efficient testing of deep neural networks via labeling-free accuracy estimation. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1776–1787. https://doi.org/10.1109/ICSE48619.2023.00152

[35] Renhao Huang, Hao Xue, Maurice Pagnucco, Flora Salim, and Yang Song. 2023. Multimodal trajectory prediction: A survey. *arXiv preprint arXiv:2302.10463* 1 (2023).

[36] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. Deepcrime: mutation testing of deep learning systems based on real faults. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*. 67–78.

[37] Nargiz Humbatova, Jinhan Kim, Gunel Jahangirova, Shin Yoo, and Paolo Tonella. 2024. An Empirical Study of Fault Localisation Techniques for Deep Learning. *arXiv preprint arXiv:2412.11304* (2024).

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

[39] Huatao Jiang, Lin Chang, Qing Li, and Dapeng Chen. 2019. Trajectory prediction of vehicles based on deep learning. In *2019 4th International Conference on Intelligent Transportation Engineering (ICITE)*. IEEE, 190–195.

[40] Jiaming Jiang, Kai Yan, Xindong Xia, and Biao Yang. 2025. A Survey of Deep Learning-Based Pedestrian Trajectory Prediction: Challenges and Solutions. *Sensors (Basel, Switzerland)* 25, 3 (2025), 957.

[41] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[42] Oliver Kramer. 2013. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*. Springer, 13–23.

[43] Yuheng Kuang, Zhengning Wang, Jianping Zhang, Zhenyu Shi, and Yuding Zhang. 2025. DA-STGCN: 4D Trajectory Prediction Based on Spatiotemporal Feature Extraction. *arXiv preprint arXiv:2503.04823* (2025).

[44] Michael P LaValley. 2008. Logistic regression. *Circulation* 117, 18 (2008), 2395–2399.

[45] Huanhuan Li, Hang Jiao, and Zaili Yang. 2023. AIS data-driven ship trajectory prediction modelling and analysis based on machine learning and deep learning methods. *Transportation Research Part E: Logistics and Transportation Review* 175 (2023), 103152.

[46] Huanhuan Li, Hang Jiao, and Zaili Yang. 2023. Ship trajectory prediction based on machine learning and deep learning: A systematic review and methods analysis. *Engineering Applications of Artificial Intelligence* 126 (2023), 107062.

[47] Yinghua Li, Xueqi Dang, Lei Ma, Jacques Klein, and Tegawendé F Bissyandé. 2024. Prioritizing test cases for deep learning-based video classifiers. *Empirical Software Engineering* 29, 5 (2024), 111.

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

[49] Yinghua Li, Xueqi Dang, Lei Ma, Jacques Klein, Yves Le Traon, and Tegawende F Bissyande. 2024. Test input prioritization for 3d point clouds. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 1–44.

[50] Yinghua Li, Xueqi Dang, Weiguo Pian, Andrew Habib, Jacques Klein, and Tegawendé Bissyandé. 2024. Test Input Prioritization for Graph Neural Networks. *IEEE Transactions on Software Engineering* (2024).

[51] Yuhao Li, Qing Yu, and Zhisen Yang. 2024. Vessel trajectory prediction for enhanced maritime navigation safety: A novel hybrid methodology. *Journal of Marine Science and Engineering* 12, 8 (2024), 1351.

[52] Zheng Li, Mark Harman, and Robert M Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering* 33, 4 (2007), 225–237.

[53] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. 2019. Boosting operational dnn testing efficiency through conditioning. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 499–509. https://doi.org/10.1145/3338906.3338930

[54] Maohan Liang, Ryan Wen Liu, Yang Zhan, Huanhuan Li, Fenghua Zhu, and Fei-Yue Wang. 2022. Fine-grained vessel traffic flow prediction with a spatio-temporal multigraph convolutional network. *IEEE Transactions on Intelligent Transportation Systems* 23, 12 (2022), 23694–23707.

[55] Siyuan Lin, Yufei Jiang, Feng Hong, Lixiang Xu, Haiguang Huang, and Bin Wang. 2025. HDFormer: A transformer-based model for fishing vessel trajectory prediction via multi-source data fusion. *Ocean Engineering* 320 (2025), 120309.

[56] Zhao Liu, Weipeng Zuo, Hua Shi, Wanli Chen, Xiao Lang, Wengang Mao, and Mingyang Zhang. 2025. An Ensemble Decision Trees Model to Predict Traffic Pattern for Maritime Traffic Management. *IET Intelligent Transport Systems* 19, 1 (2025), e70049.

[57] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 120–131.

[58] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–22.

[59] Nadya Abdel Madjid, Abdulrahman Ahmad, Murad Mebrahtu, Yousef Babaa, Abdelmoamen Nasser, Sumbal Malik, Bilal Hassan, Naoufel Werghi, Jorge Dias, and Majid Khonji. 2025. Trajectory prediction for autonomous driving: Progress, limitations, and future directions. *arXiv preprint arXiv:2503.03262* (2025).

[60] Kane Meissel and Esther S Yao. 2024. Using Cliff's delta as a non-parametric effect size measure: an accessible web app and R tutorial. *Practical Assessment, Research, and Evaluation* 29, 1 (2024).

[61] Asif Nawaz, Zhiqiu Huang, Senzhang Wang, Azeem Akbar, Hussain AlSalman, and Abdu Gumaei. 2020. GPS trajectory completion using end-to-end bidirectional convolutional recurrent encoder-decoder architecture with attention mechanism. *Sensors* 20, 18 (2020), 5143.

[62] Rongqi Pan, Mojtaba Bagherzadeh, Taher A Ghaleb, and Lionel Briand. 2022. Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering* 27, 2 (2022), 29.

[63] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18. https://doi.org/10.1145/3132747.3132785

[64] Nick Pepper, Marc Thomas, and Zack Xuereb Conti. 2026. Fast Surrogate Models for Adaptive Aircraft Trajectory Prediction in En route Airspace. In *AIAA SCITECH 2026 Forum*. 1611.

[65] Aleksandar Petrovic, Robertas Damaševičius, Luka Jovanovic, Ana Toskovic, Vladimir Simic, Nebojsa Bacanin, Miodrag Zivkovic, and Petar Spalević. 2023. Marine vessel classification and multivariate trajectories forecasting using metaheuristics-optimized extreme gradient boosting and recurrent neural networks. *Applied Sciences* 13, 16 (2023), 9181.

[66] Jakaria Rabbi, Md Tahmid Hasan Fuad, and Md Abdul Awal. 2021. Human activity analysis and recognition from smartphones using machine learning techniques. *arXiv preprint arXiv:2103.16490* (2021).

[67] H Rong, AP Teixeira, and C Guedes Soares. 2019. Ship trajectory uncertainty prediction based on a Gaussian Process model. *Ocean Engineering* 182 (2019), 499–511.

[68] Andrey Rudenko, Luigi Palmieri, Michael Herman, Kris M Kitani, Dariu M Gavrila, and Kai O Arras. 2020. Human motion trajectory prediction: A survey. *The International Journal of Robotics Research* 39, 8 (2020), 895–935.

[69] Aizaz Sharif, Dusica Marijan, and Marius Liaaen. 2021. Deeporder: Deep learning for test case prioritization in continuous integration testing. In *2021 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 525–534.

[70] Ying Shi, Beibei Yin, and Jing-Ao Shi. 2025. Markov model based coverage testing of deep learning software systems. *Information and Software Technology* 179 (2025), 107628. https://doi.org/10.1016/j.infsof.2024.107628

[71] Ying Shi, Beibei Yin, Zheng Zheng, and Tiancheng Li. 2021. An empirical study on test case prioritization metrics for deep neural networks. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 157–166.

[72] Donghwan Shin, Shin Yoo, Mike Papadakis, and Doo-Hwan Bae. 2019. Empirical evaluation of mutation-based test case prioritization techniques. *Software Testing, Verification and Reliability* 29, 1-2 (2019), e1695.

[73] Yuyol Shin, Namwoo Kim, Hyeyeong Lee, Soh Young In, Mark Hansen, and Yoonjin Yoon. 2024. Deep learning framework for vessel trajectory prediction using auxiliary tasks and convolutional networks. *Engineering Applications of Artificial Intelligence* 132 (2024), 107936.

[74] Apoorv Singh. 2023. Trajectory-prediction with vision: A survey. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3318–3323.

[75] Yan-Yan Song and Ying Lu. 2015. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry* (2015).

[76] Weifeng Sun, Meng Yan, Zhongxin Liu, and David Lo. 2023. Robust test selection for deep neural networks. *IEEE Transactions on Software Engineering* 49, 12 (2023), 5250–5278.

[77] Weizhen Tang and Jie Dai. 2025. 4D trajectory prediction for inbound flights. *Frontiers in Neurorobotics* 19 (2025), 1625074.

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

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

[80] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. 2019. Deep learning for sensor-based activity recognition: A survey. *Pattern recognition letters* 119 (2019), 3–11.

[81] Qingfan Wang, Dongyang Xu, Gaoyuan Kuang, Chen Lv, Shengbo Eben Li, and Bingbing Nie. 2025. Risk-Aware Vehicle Trajectory Prediction Under Safety-Critical Scenarios. *IEEE Transactions on Intelligent Transportation Systems* (2025).

[82] Weizhuo Wang, Michael Raitor, Steve Collins, C Karen Liu, and Monroe Kennedy III. 2023. Trajectory and sway prediction towards fall prevention. In *IEEE International Conference on Robotics and Automation: ICRA:[proceedings]. IEEE International Conference on Robotics and Automation*, Vol. 2023. 10483.

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

[84] Michael Weiss and Paolo Tonella. 2022. Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study). In *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*. 139–150.

[85] Robert F Woolson. 2007. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials* (2007), 1–3.

[86] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. Npc: N euron p ath c overage via characterizing decision logic of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27. https://doi.org/10.1145/3490489

[87] Dong Yang, Xiaoyu Li, and Lingye Zhang. 2024. A novel vessel trajectory feature engineering for fishing vessel behavior identification. *Ocean Engineering* 310 (2024), 118677.

[88] Yang Yang, Fallah Mohammadzadeh, Mohammad Khishe, Ahmed Najat Ahmed, Mosleh M Abualhaj, and Taher M Ghazal. 2025. Deep learning for sports motion recognition with a high-precision framework for performance enhancement. *Scientific Reports* 15, 1 (2025), 38861.

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

[90] Jing Yu, Shukai Duan, and Xiaojun Ye. 2022. A white-box testing for deep neural networks based on neuron coverage. *IEEE transactions on neural networks and learning systems* 34, 11 (2022), 9185–9197. https://doi.org/10.1109/TNNLS.2022.3156620

[91] Umar Zaman, Junaid Khan, Eunkyu Lee, Awatef Salim Balobaid, RY Aburasain, and Kyungsup Kim. 2024. Deep learning innovations in South Korean maritime navigation: Enhancing vessel trajectories prediction with AIS data. *PLoS One* 19, 10 (2024), e0310385.

[92] Weili Zeng, Xiao Chu, Zhengfeng Xu, Yan Liu, and Zhibin Quan. 2022. Aircraft 4D trajectory prediction in civil aviation: A review. *Aerospace* 9, 2 (2022), 91.

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

[94] Minglong Zhang, Liang Huang, Yuanqiao Wen, Jinfen Zhang, Yamin Huang, and Man Zhu. 2022. Short-term trajectory prediction of maritime vessel using k-nearest neighbor points. *Journal of Marine Science and Engineering* 10, 12 (2022), 1939.

[95] Quanjun Zhang, Chunrong Fang, Weisong Sun, Shengcheng Yu, Yutao Xu, and Yulei Liu. 2022. Test case prioritization using partial attention. *Journal of Systems and Software* 192 (2022), 111419.

[96] Xiaocai Zhang, Xiuju Fu, Zhe Xiao, Haiyan Xu, and Zheng Qin. 2022. Vessel trajectory prediction in maritime transportation: Current approaches and beyond. *IEEE Transactions on Intelligent Transportation Systems* 23, 11 (2022), 19980–19998.

[97] Chunyu Zhao, Yanzhou Mu, Xiang Chen, Jingke Zhao, Xiaolin Ju, and Gan Wang. 2022. Can test input selection methods for deep neural network guarantee test diversity? A large-scale empirical study. *Information and Software Technology* 150 (2022), 106982.

[98] Ming Zheng, Fei Wang, Xiaowen Hu, Yuhao Miao, Huo Cao, and Mingjing Tang. 2022. A method for analyzing the performance impact of imbalanced binary data on machine learning models. *Axioms* 11, 11 (2022), 607.

[99] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *Proceedings of the 10th international conference on Ubiquitous computing*. 312–321.

[100] Jianyi Zhou, Feng Li, Jinhao Dong, Hongyu Zhang, and Dan Hao. 2020. Cost-effective testing of a deep learning model through input reduction. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 289–300.

[101] Silin Zhou, Jing Li, Hao Wang, Shuo Shang, and Peng Han. 2023. GRLSTM: trajectory similarity computation with graph-based residual LSTM. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 4972–4980.