# Test Input Prioritization for Graph Neural Networks

Yinghua Li, Xueqi Dang, Weiguo Pian, Andrew Habib, Jacques Klein, *Member, IEEE,* and Tegawendé F. Bissyandé

***Abstract*—GNNs have shown remarkable performance in a variety of classification tasks. The reliability of GNN models needs to be thoroughly validated before their deployment to ensure their accurate functioning. Therefore, effective testing is essential for identifying vulnerabilities in GNN models. However, given the complexity and size of graph-structured data, the cost of manual labelling of GNN test inputs can be prohibitively high for real-world use cases. Although several approaches have been proposed in the general domain of Deep Neural Network (DNN) testing to alleviate this labelling cost issue, these approaches are not suitable for GNNs because they do not account for the interdependence between GNN test inputs, which is crucial for GNN inference. In this paper, we propose NodeRank, a novel test prioritization approach specifically for GNNs, guided by ensemble learning-based mutation analysis. Inspired by traditional mutation testing, where specific operators are applied to mutate code statements to identify whether provided test cases reveal faults, NodeRank operates on a crucial premise: If a test input (node) can kill many mutated models and produce different prediction results with many mutated inputs, this input is considered more likely to be misclassified by the GNN model and should be prioritized higher. Through prioritization, these potentially misclassified inputs can be identified earlier with limited manual labeling cost. NodeRank introduces mutation operators suitable for GNNs, focusing on three key aspects: the graph structure, the features of the graph nodes, and the GNN model itself. NodeRank generates mutants and compares their predictions against that of the initial test inputs. Based on the comparison results, a mutation feature vector is generated for each test input and used as the input to ranking models for test prioritization. Leveraging ensemble learning techniques, NodeRank combines the prediction results of the base ranking models and produces a misclassification score for each test input, which can indicate the likelihood of this input being misclassified. NodeRank sorts all the test inputs based on their scores in descending order. To evaluate NodeRank, we build 124 GNN subjects (i.e., a pair of dataset and GNN model), incorporating both natural and adversarial contexts. Our results demonstrate that NodeRank outperforms all the compared test prioritization approaches in terms of both APFD and PFD, which are widely-adopted metrics in this field. Specifically, NodeRank achieves an average improvement of between 4.41% and 58.11% on original datasets and between 4.96% and 62.15% on adversarial datasets.**

***Index Terms*—Test input prioritization, graph neural networks, mutation analysis, learning to rank, labelling.**

Manuscript received 12 May 2023; revised 8 March 2024; accepted 31 March 2024. Date of publication 5 April 2024; date of current version 14 June 2024. This work was supported by the European Research Council (ERC) through the European Union's Horizon 2020 research and innovation program under Grant Agreement 949014. Recommended for acceptance by T. Menzies. *(Corresponding author: Xueqi Dang.)*

The authors are with the University of Luxembourg, 1359 Kirchberg, Luxembourg (e-mail: yinghua.li@uni.lu; xueqi.dang@uni.lu; weiguo.pian@uni.lu; andrew.a.habib@gmail.com; jacques.klein@uni.lu; tegawende.bissyande@uni.lu).

Digital Object Identifier 10.1109/TSE.2024.3385538

## I. Introduction

RECENT years have witnessed widespread adoption of graph machine learning for modeling, predictive, and analytics tasks on graph-structured data, while the emergence of Graph Neural Networks (GNNs) [1] has led to the achievement of the unprecedented performance of a variety of applications in drug design [2], [3], [4], recommender systems [5], [6], and social network analysis [7], [8]. As they are increasingly adopted, the debugging of GNNs becomes essential, especially in safety-critical and security-sensitive domains. A key perspective in that domain is developing effective and efficient techniques for GNN testing to achieve quality assurance.

Unfortunately, Deep Neural Networks (DNNs), including GNNs, are notoriously difficult to test due to the limitations in the availability of a test oracle [9], [10], [11]. Indeed, DNN testing is challenged by the fact that it is costly and time-consuming to label test inputs: 1) automated labeling is not yet mainstream; 2) datasets can be substantially large, and the data can be complex, as in the case of GNNs; 3) labeling may require deep domain-specific knowledge, which is prohibitively expensive to acquire. Therefore, to achieve efficient and effective testing of DNN-based systems, researchers and practitioners generally focus on identifying only the relevant test inputs that are likely to cause the system to behave incorrectly (i.e., bug-revealing test inputs). Diagnosing those inputs is then expected to provide insights for debugging the DNNs.

Prior work has developed various techniques to identify and prioritize bug-revealing test inputs, which allows testers/developers to focus on the most critical inputs [10], [11], [12], [13]. Such test prioritization techniques aim at optimizing the time as well as the required resources for testing. A large majority of DNN test prioritization approaches fall within three categories [11]: coverage-based, confidence-based and surprise-based approaches. Confidence-based approaches, such as DeepGini [10], prioritize test inputs based on model confidence: a test input is more likely to be incorrectly predicted via a DNN model if that model outputs similar prediction probabilities for each class. Coverage-based approaches, such as CTM [14], simply adapt coverage-based test prioritization from traditional software systems testing into DNN testing and have been shown to underperform against confidence-based approaches [10]. Surprise-based methods [13], [15] perform test prioritization based on the surprise of test inputs. This "surprise" is quantified by measuring the distance in neuron activation patterns between a test input and the training data. However, existing studies [16] have demonstrated that surprise-based methods are less effective than confidence-based approaches. Furthermore, surprise-based methods typically come with higher computational costs due to the need for more parameter tuning.

Although confidence-based approaches have demonstrated effectiveness in the context of DNNs, they suffer from several limitations when applied to GNNs. Notably, they do not account for the interdependence inherent in graph-structured test inputs composed of nodes and edges. These approaches were originally designed for DNNs, where tests are independent of each other. Additionally, confidence-based approaches operate under the assumption that test inputs for which the model exhibits low confidence are more likely to be misclassified and, therefore, should be given higher priority. However, in the presence of adversarial attacks, the model's confidence can be higher for incorrect predictions, leading to erroneous outputs.

More recently, novel approaches such as PRIMA [11] are being introduced in the literature of DNN testing, leveraging techniques such as mutation analysis. However, PRIMA, the state-of-the-art in DNN test prioritization, cannot be applied to GNNs since their mutation operators are not adapted to graph-structured data and models. Dang et al. [17] proposed GraphPrior, a test prioritization method specifically designed for GNNs. Despite GraphPrior also relying on mutation analysis, there are significant differences between NodeRank and GraphPrior:

- **Incorporating Input Mutations** GraphPrior performs test prioritization solely based on model-specific mutations, whereas NodeRank not only considers model mutations but also takes into account mutations specific to the input. NodeRank considers two types of input mutations: 1) Node feature mutations, which are designed to perturb the feature attributes of selected nodes, consequently influencing the representation and information flow within the graph; 2) Graph structure mutations, which aim to alter the interdependence of the test inputs within the graph by introducing additional edges, thus changing the structural properties of the graph.
- **Leveraging Ensemble Learning Techniques for learning-to-rank** In contrast to GraphPrior, which employs a single ranking model to learn the misclassification probability of test inputs, NodeRank leverages ensemble learning techniques to integrate multiple base ranking models with the aim of optimizing its performance. Existing studies [18], [19], [20] have demonstrated that ensemble learning typically achieves higher accuracy than single ML models. Furthermore, our analysis delves into the influence of different ensemble techniques on NodeRank and illustrates that the sum-based ensemble technique yields the best performance.
- **Considering Different Killing Methods** GraphPrior simply assumes that a mutated model is considered "killed" if the predictions of the original model and the mutated model for the test input differ. However, prior research [21] has highlighted that in the context of DNN mutation analysis, variations in the outputs between a mutated model and the original model can occur solely due to the inherent randomness in the training process rather than because the mutant is actually discriminated from the original model. Therefore, we utilized the killing method provided by DeepCrime [21] for test prioritization and generated relevant variants of NodeRank. In DeepCrime, the killing process involves iteratively training both the original model and the mutated model, then comparing the distribution difference in their outputs to determine whether the mutated model is "killed." This approach can contribute to mitigating the impact of randomness in the training process. Based on the DeepCrime approach, by comparing NodeRank variants utilizing model mutation rules and those not utilizing model mutation rules, we demonstrated that mutations generated by the model mutation rules of NodeRank contribute to its effectiveness.

**This paper.** We propose NodeRank (**Node Rank**ing for graph-structure test inputs), a novel test input prioritization approach targeting GNNs. NodeRank leverages the ideas from traditional mutation testing [22], [23] to prioritize potentially misclassified test inputs so that such tests can be identified earlier with limited manual labeling costs. More specifically, the core idea of NodeRank is that: a test is considered more likely to be misclassified if this test can kill many mutated models and produce different prediction results with many mutated inputs.

NodeRank is a test prioritization approach that is model-based, input-based, and mutation testing-based. It applies mutation operations to GNN models and tests, generating mutation features for test prioritization. Specific mutation operations applied are described below.

In NodeRank, we developed three distinct types of mutations, namely graph structure mutation (GSM), node feature mutation (NFM), and GNN model mutation (GMM), based on the characteristics of GNNs and the graph test dataset. GSM aims to modify the interdependence of the graph test inputs by introducing additional edges, thereby altering the structural properties of the graph. NFM, on the other hand, perturbs the feature attributes of selected nodes, thereby influencing the representation and information flow within the graph. Both GSM and NFM can be categorized as input mutations as they directly modify the characteristics of the dataset. In contrast, GMM is specifically developed to mutate GNN models, with the objective of modifying the message passing of the GNNs by changing specific training parameters. The GMM mutation type thus falls under the category of model mutations.

For each test input, NodeRank generates these three types of mutations, as described above. Subsequently, by comparing the prediction results before and after the mutation, NodeRank generates a mutation feature vector for each test. Specifically, for graph input mutation (i.e., GSM and NFM), if a mutated input fails (i.e., the predictions for the mutated inputs and the original inputs are different), the corresponding element in the relative feature vector is marked as 1; otherwise, it is marked as 0. For GNN model mutation (i.e., GMM), if a mutated model is killed (i.e., the prediction results for this input via the mutated models and the original models are different), the corresponding element in the relative feature vector is marked as 1; otherwise, it is marked as 0. The mutation feature vector of each test is then fed into pre-trained ranking models, which are designed to predict the likelihood of this input being misclassified. Our ranking models are trained to automatically predict a misclassification score indicating its likelihood of being misclassified by the model.

To further enhance the performance of our ranking models, we adopt ensemble learning techniques that combine the predictions from multiple base ranking models. The idea draws inspiration from the field of ensemble learning [18], [19], [24], which aims to improve overall performance by integrating predictions from two or more base machine learning models. Notably, ensemble learning methods have achieved state-of-the-art outcomes across various machine learning applications [25], [26], [27], [28]. In the NodeRank framework, we employ three distinct ensemble methods [18], [29], [30] to effectively combine the outputs of the individual ranking models.

It is important to note that, NodeRank differs from the state-of-the-art test prioritization approach, PRIMA, in several domains: the **target** (GNN vs. DNN) as well as the **approach** (mutation rules and ranking strategies).

- **Target.** NodeRank is designed to address the test prioritization problem in GNNs and, therefore, operates on datasets that exhibit complex interdependence between individual test inputs. In contrast, PRIMA is intended for traditional DNNs, where each sample in the dataset is independent.
- **Mutation rules.** NodeRank's mutation rules can affect the interdependency between test inputs from two perspectives: First, NodeRank's model mutation rules can directly or indirectly affect the message passing between nodes in graph data. More specifically, in the mutated GNN model, the manner in which nodes acquire information from their neighboring nodes is slightly different from that of the original GNN model. Second, NodeRank's node mutation rules modify the interdependence between nodes by adding edges to nodes. When adding a new edge from node A to node B, a new connection is built, and the prediction of node A is now impacted by the newly connected node B, thus changing the node interdependence. In contrast, the mutation rules of PRIMA are specifically designed for independent test inputs and, therefore, do not impact the relationships between tests.
- **Ranking strategies.** NodeRank leverages ensemble ranking models to learn from mutation results for test prioritization. These models are constructed by combining different base ranking models, thereby improving the overall performance of the model [18], [19], [24]. In contrast, PRIMA employs a single ranking model for test prioritization.

We evaluate the performance of NodeRank based on 124 subjects (i.e., a pair of dataset and GNN model). Our evaluation considers both natural inputs and graph adversarial inputs, which are generated by eight graph adversarial attacks [31], [32], [33], [34]. We compare NodeRank with multiple test prioritization approaches. Our experimental results demonstrate that, on natural datasets, the average improvement of NodeRank over the compared approaches, in terms of APFD, is between 4.41% and 58.11%. On graph adversarial inputs, the average improvement of NodeRank over the compared approaches in terms of APFD ranges from 4.96% and 62.15%.

NodeRank can be applied across diverse real-world contexts. For instance, a typical use case of node classification in GNNs is fraud detection [35] in banking transfer transaction systems. Here, each account can be represented as a node, while the transactional interactions between them can be represented as edges. Through node classification, these accounts can be categorized as normal or fraudulent. When developers use GNNs to predict whether each node (account) is a fraudulent account, the GNNs can exhibit wrong prediction behavior, such as predicting fraudulent accounts as normal accounts, which can lead to losses for the bank. In this scenario, NodeRank can be utilized to prioritize potentially misclassified accounts (with those more likely to be misclassified ranked at the top). These sorted accounts can be provided to bank staff, allowing them to quickly perform manual checks on the accounts that are more likely to be misclassified, thus reducing losses.

The contributions of this paper are as follows:

- **Approach.** We propose a novel approach, NodeRank, to prioritize test inputs for GNN models. NodeRank introduces three distinct types of mutation rules that target the mutation of graph structure, node features, and GNN models, respectively and adopt ensemble-learning-based learning-to-rank to intelligently combine mutation results for effective test input prioritization.
- **Study.** We conducted a large-scale study based on 124 subjects to evaluate the effectiveness of NodeRank on both natural and adversarial inputs. The experimental results demonstrate its effectiveness.
- **Performance Analysis.** We provide an extensive analysis of the performance of NodeRank by investigating the influence of the different ensemble learning strategies as well as by performing an ablation study to showcase the contributions of the different mutation feature sets.

Our dataset, code, and results are made publicly available in a replication package[1] for the community.

## II. Background

We now briefly introduce the key domain concepts for our work.

### A. Graph Neural Networks

Graph neural networks [36], [37], [38] have achieved great success in solving machine learning problems on graph-structured data [7], [39], [40]. Initial models learned

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

representations of target nodes by propagating neighborhood information through recurrent neural architectures in an iterative manner until a stable fixed point is reached. Subsequently, several variations have been proposed in the literature: Kipf et al. [41] proposed Graph convolutional networks (GCN), which adapt convolution techniques from classical convolutional neural networks, to graph data. GCN implements message passing of multi-order neighborhoods by superimposing several convolutional layers. More recently, other GNN architectures have been proposed towards taking into account the advancements in the field of deep learning: for example, Veličković et al. [38] proposed graph attention networks (GAT) which uses attention techniques to assign different weights according to the importance of nodes in the graph.

In GNNs, a graph is usually defined as a data structure composed of nodes and edges. We denote a graph as $G = (V, E)$ where $V = \{1, 2, \ldots, N\}$ refers to the set of N nodes, and $E \subseteq V \times V$ refers to the set of edges. In GNN datasets like Cora (a node classification dataset), each node represents a scientific paper, while edges represent citation relationships between papers. In this dataset, test inputs typically refer to new nodes (scientific papers) that have not been seen during the training process. In the case of the Cora dataset, given a test input (a scientific paper), a GNN model is used to classify the paper into specific categories. In other words, the GNN model predicts the categories that best describe the content of the given paper. For instance, these categories can be "reinforcement learning" and "neural networks," implying that the paper belongs to the "reinforcement learning" or "neural networks" category.

**[GNN training process]** GNNs undergo a training process similar to other neural networks. The inputs required for GNN training typically include: 1) **Graph Structure.** Graph structure information encompasses the connections between nodes in the graph; 2) **Node Features.** Each node typically comes with associated feature vectors, which reflect the attributes of the node; 3) **Target Labels.** In the training data for GNN node classification, "Target Labels" refer to the category to which each node belongs. These labels are typically predefined.

During the training process of GNNs, several components are continually trained and optimized: 1) **Model Parameters.** The primary aim of GNN training is to refine the model parameters. These parameters include weights and biases linked to operations like graph convolutions and aggregation functions within the GNN architecture. 2) **Node Embeddings.** GNNs comprise layers with associated parameters, and part of the training process involves learning these node embeddings. Node embeddings are vector representations of individual nodes within the graph. They capture a node's structural and feature-based information and evolve as the model trains; 3) **Loss Function.** The loss function plays a pivotal role in GNN training. It quantifies the disparity between the model's predictions, typically pertaining to nodes or graph-level attributes, and the actual ground truth labels for the given task. Throughout training, model parameters are iteratively adjusted to minimize this loss function.

The specific training process for GNNs typically consists of the following steps:

- **Initialization**: All GNN parameters are randomly initialized, typically with small random values.
- **Forward Propagation**: For each node, its node embedding is updated based on the information from its neighbors. This is typically achieved using weight matrices and aggregation functions (e.g., mean or max pooling). This aggregation process can go through multiple layers, allowing information to propagate further in the graph.
- **Loss Calculation**: This process calculates the loss value based on the GNN's output and the true labels.
- **Backpropagation**: This process computes the gradients of the loss function with respect to each parameter.
- **Parameter Weight Updates**: This process updates each parameter weight based on the gradient values.
- **Iterate**: This process repeats the above steps (forward propagation, loss calculation, backpropagation, parameter updates) until a stopping condition is met, such as reaching a predefined number of iterations.

It is important to note that graphs used in GNN training differ from normal graphs. Specifically, GNN training graphs include feature attributes for nodes. Furthermore, in tasks like node classification, nodes in the graphs have category labels. In contrast, normal graphs usually comprise only the topological structure of nodes and edges, without specific labels or node attributes.

**[GNN inference]** In the context of GNNs, *inference* refers to using a pre-trained GNN model to perform prediction on new graph data. For example, in node classification tasks, the GNN model utilizes its learned parameters and weights to classify a given node. The input typically consists of the features of the node and the graph structure of its belonged graph. The output is the classification of the node.

## B. Mutation Testing

Mutation testing [22] is a software testing method that aims to evaluate the quality of the test suite by intentionally introducing small changes (called mutations) into the source code and observing the test suite's reaction. The core objective is to determine the effectiveness of test suites in finding code bugs. The intuition is that if a test case can detect intentionally-introduced errors, this test case is more likely to detect real bugs in practice. Mutation testing has achieved state-of-the-art performance by providing a comprehensive evaluation of the test suite via creating and testing multiple variations (mutants) of the code, ensuring that the test cases are thoroughly covering different scenarios and even edge cases, which is difficult to achieve through traditional testing methods.

In mutation testing, *kill* and *fail* are terms used to describe the results of running a test suite on a set of artificially created code changes or 'mutants' to evaluate the quality of the test suite. Specifically, a mutant is regarded as 'killed' if the behavior of this mutant differs from that of the original code, indicating that the test suite is capable of detecting the fault introduced by the mutant. A test input is said to 'fail' if it is not passed by the target program.

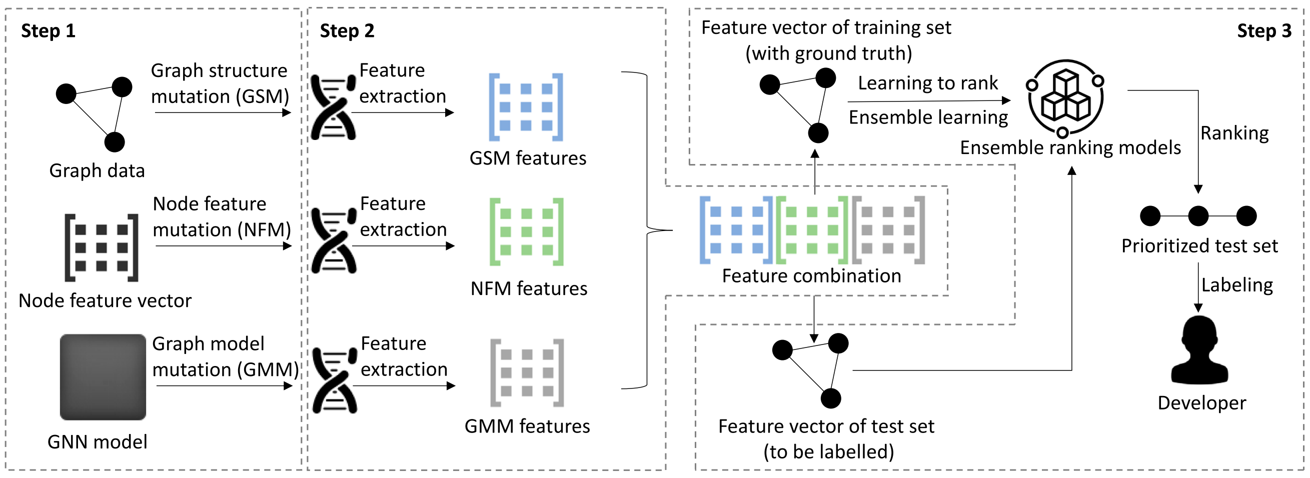


Fig. 1. Overview of NodeRank.

### C. Ensemble Learning

Ensemble learning [18] is a meta approach in machine learning where multiple, generally diversified, ML models are combined to achieve better performance and generalization. Common examples of ensemble learning strategies include Majority voting and Stacking. Majority voting is a straightforward approach that sums for each prediction class the number of yielded predictions by the different models: the class with the majority number of predictions is then outputted by the ensemble model. On the other hand, Stacking utilizes a meta-model such as logistic regression to learn how to optimally combine the predictions from base ML models.

Ranking is crucial to many real-world applications, notably in the field of information retrieval. In software engineering, test prioritization assumes the possibility to rank test cases according to their ability to reveal faults. In recent studies [18], the ranking has been formalized as a machine learning problem, and *ensemble ranking* often employs ensemble learning techniques to learn optimal weights for combining multiple ranking algorithms.

## III. Approach

### A. Overview

NodeRank is a model-based, input-based, and mutation testing-based test prioritization approach. By employing mutation operations on both GNN models and test inputs, NodeRank produces mutation features for each test input and predicts the misclassification probability of the input in order to perform test prioritization. Fig. 1 presents the overview of the different steps in our NodeRank test prioritization approach. First, we offer detailed explanations for certain elements in Fig. 1 and provide reasons for the symbols utilized for them, with the goal of enhancing the understanding of the figure.

- The second dotted square in Step 1 represents an $N \times M$ matrix. This matrix is used to encapsulate the feature vectors of all nodes. Specifically, each row of the matrix represents a node's feature vector. There are $N$ rows in the matrix corresponding to the $N$ nodes in the dataset. $M$ columns represent that each node has $M$ features.
- The reason we use a dotted representation for node feature vectors and mutation features is that, in our experiments, these features are both represented using matrices. The dotted square can serve as a visual abstraction of a matrix. A matrix comprises multiple values, and we use dots to represent the values within the matrix abstractly. For example, in the second dotted square in Step 1, the third dot in the second row represents the value of the third feature of the second node in the graph dataset.
- It is important to note that the meaning of the dotted squares in Step 1 and Step 2 is different. However, since they both represent matrices, we use dotted squares with different colors to distinguish them. In Step 1, the dotted square represents the node feature vector, while in Step 2, the dotted square represents mutation features generated from the mutation results.
- We utilize arrows to illustrate processes and operations. For instance, in Step 1, the Graph data undergoes graph structure mutation within Step 1 and feature generation in Step 2, resulting in graph input mutation features. Another example involves the Node feature vector in Step 1, which undergoes node feature mutation in Step 1 and feature generation in Step 2 to yield node mutation features.
- The chromosome symbols represent mutation results. In Fig. 1, a chromosome with a break indicates the mutations applied to the GNN model or inputs, resulting in mutation results. These mutation results are subsequently used for mutation feature generation in order to perform test prioritization.

Moreover, each box represents a step in the NodeRank workflow. In the following section, we offer a general overview of each step, as depicted in Fig. 1. Specific details for each step can be found from Section III-B to Section III-D.

❶ *Generating mutants.* NodeRank generates mutants for three different inputs: the graph structure itself (which represents the interdependence of samples in the datasets), the node features (which represents sample data), and the GNN model (which is learned and is the target of testing). To that end, we develop specific mutation rules that are carefully designed for GNN testing. Section III-B details those rules that must be applied to generate mutants for a given test set $T$, the graph structure $G$ of the data, and the GNN model $M$ under test.

❷ *Extracting and combining mutation features.* NodeRank then obtains the model prediction towards the mutants and the original test inputs. By comparing the predictions, NodeRank generates the mutation feature vector for each input. The detailed description is as follows. Given $M'$, a mutant of $M$, NodeRank considers that a test input kills $M'$ if the prediction on this test input by $M'$ is different from the prediction by $M$. For a given mutant of a test input $t \in T$, NodeRank considers that this mutant failed if it leads to a prediction that is different from the prediction using $t$. Given $G'$, a mutant of $G$, NodeRank considers that the mutant fails if the prediction of the GNN using $G'$ is different from its prediction when using $G$.

Based on the execution outputs, NodeRank builds feature vectors to train a ranking model. These are referred to as **mutation features** and are of three types: Node mutation features, graph structure mutation features, and model mutation features (cf. Section III-C for details).

❸ *Ranking test inputs using ensemble ranking models.* Eventually, for each test input, NodeRank produces three vectors, which represent three types of mutation features. These vectors are then concatenated to produce a mutation feature vector $v$ for each test input $t \in T$. Given all test inputs from $T$, NodeRank, therefore, leverages ensemble ranking models based on their associated mutation features to predict ranking scores of the test inputs. These scores, ordered in a descending way, are used to prioritize the associated test inputs accordingly.

The findings presented in Section V provide compelling evidence for the effectiveness of NodeRank, which can be attributed, in part, to the careful design of mutation rules and the effective ensemble strategy of ranking models. 1) Our designed mutation rules can effectively generate informative mutation features by leveraging the interdependence of test inputs. The node mutation rules operate by introducing new edges between nodes in the graph dataset, which impacts the interdependence structure of the data. The model mutation rules affect message passing between nodes in the GNN prediction process, leading to small changes in node interdependence. 2) NodeRank adopts an ensemble ranking model for test prioritization, which leverages the strengths of multiple base ranking models to improve the overall performance. By comparing different ensemble strategies, we are able to identify the most suitable approach for use in NodeRank's test prioritization process.

In the remainder of the section, we will describe in detail the mutation rules that we have designed for NodeRank (cf. Section III-B), the construction process of the mutation feature vectors (cf. Section III-C), the setup of the ensemble ranking model (cf. Section III-D) and the application of NodeRank (cf. Section III-E).

## B. Specifying Mutation Rules

We design mutation rules that are adapted to the key main ingredients of a GNN: the graph structure of the data, the nodes in the graph, and the GNN model itself, which are explained in detail as follows.

*1) Graph Structure Mutation (GSM):* Graph structure mutation is designed to introduce slight changes to the input graph by randomly incorporating new edges. Consequently, when provided with a test input node, denoted as $t \in T$, we create mutants by adding one or more edges between node $t$ and a randomly selected node, denoted as $s \in T$. For a given node $t \in T$, the following mathematical formula provides an intuitive representation of the GSM mutation:

$$G' = G + \sum_{i=1}^{n} \mathrm{addEdge}\left(G, t, s_i\right) \tag{1}$$

where $G$ represents the original graph. $G'$ represents the mutated graph structure. In each iteration, we use the *addEdge* function to generate an edge from node $t$ to a randomly selected node $s_i \in T$. We use the symbol "+" to denote the addition of the newly generated edge to the original graph $G$. This process is repeated $n$ times, resulting in the addition of $n$ edges to the original graph $G$.

*2) Node Feature Mutation (NFM):* Given a test set and the features of the test inputs, node feature mutation aims to slightly change the features of the targeted nodes in order to offset their position in the feature space. This offset implies the modification of feature values in the different dimensions.

In the following, we introduce how node feature mutation is performed in detail. Given the original test set $T$, which consists of $n$ nodes, each node $t$ is characterized by $m$ dimensions, where each dimension corresponds to a specific feature value of the node $n$. In this case, $T$ can be represented as an $n \times m$ feature matrix. To perform node feature mutation, we apply an offset to this matrix. Specifically, assuming the degree of offset is denoted as $\alpha$, Formula 2 represents the mutation process for the test set $T$. As observed in the formula, the initial step involves multiplying the matrix of the original test set $T$ by the offset degree to calculate the ultimate offset to be applied to $T$. Subsequently, the matrix of the mutated test set, denoted as $F(T')$, is derived by adding $T$'s matrix to the offset $\alpha * F(T)$.

$$F(T') = F(T) + \alpha * F(T) \tag{2}$$

where $F\left(T'\right)$ is the feature matrix of the mutated test set $T'$, $F(T)$ is the feature matrix of the original test set $T$, and $\alpha$ is the coefficient of the degree of offset.

*3) GNN Model Mutation (GMM):* Given a trained graph neural network model, the GNN model mutation aims to change the training parameters slightly. Formula 3 offers an intuitive representation of the GMM mutation. For integer or float type parameters, the mutation operation involves making slight adjustments to the parameters. In the case of Boolean-type parameters, the mutation operation involves switching between True and False. Therefore, the formula is as follows:

$$M' = \begin{cases} M(\theta + \beta \cdot \theta) & \text{if } \theta \in \mathbb{R} \\ M(\neg\theta) & \text{if } \theta \in \{\text{True, False}\} \end{cases} \tag{3}$$

where $M'$ refers to the mutated GNN model. $M$ refers to the original GNN model, $\theta$ refers to a parameter of the original model $M$, and $\beta$ refers to the coefficient of change, indicating the magnitude of parameter change. The symbol $\neg$ signifies the logical negation operation, which inverts the parameter $\theta$. If the original value is True, it becomes False, and if the original value is False, it becomes True.

In NodeRank, we consider the following:

- **Learning Additive Bias (LAB)** [41], [42], [43] The LAB parameter is a Boolean variable that determines whether to introduce a predetermined offset to the representation vectors of nodes in the GNN model. By enabling the LAB parameter (set to True), a bias parameter is assigned to each node's representation vector. This allows the GNN model to capture the intrinsic properties of the graph better and improve the interdependence between nodes in the prediction process.
- **Negative Slope (NS)** [42] NS is a float parameter that controls the slope of the negative part of the activation function used in the Gated Linear Unit (GLU) operation, a commonly used non-linear function for message passing in GNNs. In particular, GLU combines the node features with the weighted sum of their neighboring nodes' features, which is the message passed between nodes in the graph. The negative slope parameter of the activation function in the GLU operation determines the rate of decrease for negative input values and can affect the message passing between nodes. As such, the value of NS plays a crucial role in determining the sensitivity of the GNN model to negative input values and the resulting impact on the interdependence between nodes in the graph.
- **Changing Multi-head Attentions (CMA)** [42] CMA is an integer type parameter that determines the number of attention heads employed by the GNN model, with an increase in CMA leading to an expanded model capacity and improved capacity to capture the interdependencies that exist among the nodes in the graph.
- **Concat (CON)** [42] The CON parameter is a Boolean-type parameter that determines the method used to integrate node embeddings from neighboring nodes. When set to True, the concatenation operation is employed to combine the node embeddings of adjacent nodes, resulting in a more sophisticated and expressive node representation. This, in turn, enhances the capacity of the GNN model to capture more interdependencies between nodes.
- **Adding Self Loops (ASL)** [41], [42] The ASL parameter is a Boolean parameter that governs the addition of self-loops to the input graph in graph neural networks. By setting ASL to 'True', self-loops are introduced to each node in the graph, enabling the aggregation of intrinsic information from nodes into their representation vectors. This operation modifies the weighting of neighboring nodes and can affect the interdependence of nodes during the prediction process.
- **Adding Layer Computations (ALC)** [41] ALC is a Boolean type parameter that determines whether or not to include additional layers of computation in the GNNs. When ALC is set to true, additional layers are introduced to the network, which allows for more complex transformations of the node features. As a result, the message passing process becomes more refined and capable of capturing more intricate dependencies among the nodes.
- **Hidden Channel (HC)** [41], [42], [43], [44] The HC parameter is an integer configuration parameter that governs the dimensionality of the hidden representation in each layer of the GNNs. As such, modifications to this parameter can impact the interdependence of nodes in a given graph by allowing the GNN to learn more expressive and informative node embeddings.

We explain how the mutation rules of NodeRank utilize node interdependence to generate mutations as follows:

- For Model-level mutants: NodeRank's mutant rules can directly or indirectly affect the message passing between nodes in graph data. More specifically, in the mutated GNN model, the manner in which nodes acquire information from their neighboring nodes is slightly different from that of the original GNN model.
- For Node-level mutants: NodeRank modifies the interdependence between nodes by adding edges to nodes. When adding a new edge from node A to node B, a new connection is built, and the prediction of node A is now impacted by the newly connected node B, thus changing the node interdependency.

Note that the mutation rules of NodeRank are specifically developed for GNNs, and its applicability in the context of DNNs has not yet been examined. Specifically, regarding node mutation rules, NodeRank focuses on modifying the connection relationships between nodes in a graph. However, in DNNs, the samples within a dataset are independent and lack any inherent connectivity, rendering the proposed mutation rules unsuitable for such datasets. Moreover, the model mutation rules of NodeRank are designed to impact the message passing between nodes during the prediction process, either directly or indirectly. In contrast, conventional DNNs generally consist of independent samples within a dataset, implying that such mutation rules are unlikely to influence the transmission of information between distinct tests.

## C. Constructing Mutation Features Vectors

Leveraging the three types of mutation rules introduced in the previous steps, we generate a mutation feature vector for each test input. To this end, we execute the three mutation rules, thereby generating three distinct feature vectors for each input. These feature vectors are concatenated to build the final mutation feature vector. In the following, we explain the generation of each feature vector of different mutation types.

**Dataset mutation (NFM and GSM)** Given a test input $t$ and a GNN model $M$, we denote the mutants of $t$ as $\{t_1, t_2, ..., t_n\}$, which are obtained using NFM mutation rules. We associate a vector $V$ of size $n$ to the test input $t$ where $n$ is the number of mutants and $V[k]$ maps to the execution output for the mutant $t_k$. *If $t_k$ fails (i.e., the prediction of $t_k$ is different from that of $t$), then $V[k]$ is set to 1. Otherwise, it is set to 0.* We use the same procedure to build a graph mutation features vector for $t$ using the GSM mutation rules. Formula 4 describes the process of dataset mutation in our mutation testing operation.

$$V[k] = \begin{cases} 1 & \text{if } M\left(t_k\right) \neq M(t) \\ 0 & \text{if } M\left(t_k\right) = M(t) \end{cases} \tag{4}$$

where $M(t_k)$ represents the prediction of the GNN model $M$ for mutant $t_k$, and $M(t)$ represents the prediction for the original test input $t$.

**Model mutation (GMM):** Given a test input $t$, a GNN model $M$ and its mutants $\{M_1, M_2,..., M_n\}$, we associate to the test input $t$, a vector $V$ of size $n$ (i.e., the number of mutants of $M$) where $V[k]$ maps to the execution output for the mutant $M_k$ with test input $t$. *If $t$ kills the mutated model $M_k$ (i.e., the prediction of $v_k$ via the original model $M$ and the mutated model $M_k$ is different), then $V[k]$ is set to 1. Otherwise, it is set to 0.* Formula 5 presents the process of model mutation in our mutation testing operation.

$$V[k] = \begin{cases} 1, & \text{if } M(t) \neq M_k(t) \\ 0, & \text{if } M(t) = M_k(t) \end{cases} \tag{5}$$

where $M(t)$ represents the prediction of the original model $M$ for the test input $t$. $M_k(t)$ represents the prediction of the mutant model $M_k$ for the same test input $t$.

## D. Building an Ensemble Ranking Model

Based on the previous step, NodeRank generates a feature vector $V_i$ for each test input $t_i \in T$. This feature vector is then used as the input to the ensemble ranking model for predicting the misclassification probability of $t_i$. The design of the ensemble ranking models is motivated by the principles of learning-to-rank [45] and ensemble learning [18]. In particular, we adopt four base ranking models, including Logistic Regression [46], Random Forest [47], XGBoost [48], and LightGBM [49], to form ensemble models that can leverage the strengths of each individual model. NodeRank uses the sum-based ensemble learning method [18], which combines scores of the base ranking models for a given test input. By inputting $V_i$ into the sum-based ensemble ranking model, NodeRank obtains a misclassification score for $t_i$, which can be used to estimate the probability that the GNN model $M$ will misclassify $t_i$.

Our experiments further consider two other ensemble learning methods (i.e., stacking-based [29] and voting-based [18]) to build variants of NodeRanks and assess the effectiveness of our design choices (cf. Section IV-F).

## E. Usage of NodeRank

The inputs of NodeRank are a test set $T$ and a GNN model $M$. The output is the prioritized test set $T^P$. NodeRank generates mutants for the test set $T$ and the GNN model $M$ and exploits the execution outputs of the GNN on these mutants to build feature vectors that can be utilized to learn to prioritize test inputs using ensemble ranking models. We present the training process of each ranking model as follows.

❶ **Dataset Split** Given a GNN model $M$ with dataset $T$, we partition the dataset $T$ into two subsets: a training set $R$ and a test set. Following common practice in the field [50], we allocate 70% of the data to the training set and consider the remaining 30% as the test set. We emphasize that the test set is kept entirely separate from the training process and is only utilized to evaluate NodeRank.

❷ **Training set construction** Based on the given training set $R$, the objective of this step is to build a training set $R'$ for training the ranking models. Firstly, for each input $r_i \in R$, three types of mutants are generated, and based on the execution of these mutants, the mutation feature vector $V_i$ of $r_i$ is obtained. Subsequently, the mutation feature vector of $r_i$ is utilized to build the features of the training set $R'$. Secondly, the original GNN model $M$ is used to classify each input $r_i \in R$ and compare it with the ground truth of $r_i$. This step helps identify whether $r_i$ is misclassified by the GNN model $M$. If $r_i$ is misclassified by $M$, it is labeled as 1, and if not, it is labeled as 0. This process aids in building the labels of the ranking model training set $R'$.

❸ **Training ranking models** After building $R'$, we train the ranking model based on it.

Notably, the training set $R'$ contains binary labels (i.e., 1 or 0), whereas the ranking models are expected to output continuous values, referred to as misclassification scores. To address this, we made certain modifications to the ranking algorithms we employed, such as the random forest. During the classification process, these algorithms calculate an intermediate value, which is used to decide whether an input belongs to a particular class. If the intermediate value exceeds a predefined threshold of 0.5 (which is configurable), the input is classified into the first class; otherwise, it is classified into the other class. Rather than outputting the binary label, we directly output the intermediate value, representing the misclassification score. This score indicates the likelihood of a test input being misclassified by the GNN model, with a higher score indicating a greater probability of misclassification.

## IV. Evaluation Design

To assess NodeRank, we enumerate various research questions (cf. Section IV-A), which explore the performance metric (cf. Section IV-B) for test inputs prioritzation on a diverse set of GNN subjects (cf. Section IV-D). Beyond the prioritization performance of NodeRank in uncovering model misclassification, we also consider the performance under adversarial settings (cf. Section IV-E). In this section, we also present how the design of the different variants of NodeRank(cf. Section IV-F), which vary based on the ensemble ranking strategy. Finally, information about implementation and configuration setup is provided in Section IV-G.

## A. Research Questions

We investigate the following research questions:

- **RQ1: What is the effectiveness of NodeRank?**
Building on studies in traditional software testing [51], [52], effective test prioritization techniques should be able to prioritize possibly-misclassified test inputs.

- **RQ2: How does NodeRank perform on adversarial inputs?**
  Graph adversarial attacks [32], [33] can induce GNN models to be confident in their however-incorrect predictions. Thus, existing confidence-based test prioritization approaches are likely to fail. We demonstrate the superior performance of NodeRank under such settings.
- **RQ3: How does NodeRank perform with different ensemble ranking strategies?**
  We investigate the performance of NodeRank variants implemented by considering three different ensemble learning techniques.
- **RQ4: Are all mutation feature categories useful in NodeRank?**
  We conduct an ablation study on NodeRank to assess the contribution of graph structure mutation features, node mutation features, and graph model mutation feature on the performance of NodeRank. Our ablation experiments follow prior work by Meyes et al. [53].
- **RQ5: Do the model mutation rules of NodeRank contribute to its effectiveness?**
  In the original NodeRank, for a given test input, we employ the killing approach from traditional mutation testing [54] to generate model mutation features. These features are then utilized to predict the misclassification probability for this input. However, the model mutation features generated by such a killing approach can contain information from both the model mutation rules and the randomness inherent in mutated model training, both of which can contribute to the effectiveness of NodeRank. In this research question, we aim to demonstrate that the model mutation rules actually contribute to the effectiveness of NodeRank by employing the killing approach in DeepCrime [21], which takes into account the training randomness of the mutated models during the killing process.
- **RQ6: How do the parameter ranges of the newly designed mutation operators impact the effectiveness of NodeRank?**
  In NodeRank, we developed a set of novel mutation operators tailored for GNNs. In this research question, we investigate how the parameter ranges of these newly designed mutation operators affect the performance of NodeRank.

## B. Performance Metric

We evaluate the effectiveness of test prioritization based on the common Average Percentage of Fault-Detection (APFD) [14] metric. Specifically, higher APFD values indicate faster misclassification detection rates. Given a GNN model $M$ under the test set $T$, the APFD values are calculated via Formula 6.

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

where $n$ is the number of test inputs in $T$; $k$ is the number of test inputs in $T$ that will be misclassified by $M$; $o_i$ represents the position of the $i_{th}$ misclassified test within the prioritized test set. When the sum of the index values for the first $k$ misclassified tests, i.e., $\sum_{i=1}^{k} o_i$, is small, it indicates that the prioritized test set has a higher order of the misclassified tests, leading to a larger APFD score. Consequently, a higher APFD score indicates better prioritization effectiveness.

Following prior work [10], we perform normalization on the APFD values, making them fall in the range of [0, 1] to facilitate comparison. We thus assume a test prioritization approach is better if its APFD value is closer to 1.

To conduct a more detailed evaluation, we employ the Percentage of Fault Detected (PFD) metric [10] to quantify the fault detection rate of each test prioritization approach across varying ratios of prioritized test inputs. High PFD values indicate higher effectiveness in identifying misclassified test inputs. PFD is calculated based on Formula 7.

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

where $F_c$ is the number of misclassified test inputs that are correctly detected. $F_t$ is the total number of misclassified test inputs.

In this study, we compare the PFD of NodeRank and the uncertainty-based test prioritization approaches against different ratios of prioritized tests. We use **PFD-n** to represent the first n% prioritized test inputs.

## C. Compared Approaches

This study utilized five compared approaches, including a baseline approach (i.e., random selection) and four DNN test prioritization techniques. The selection of these methods was driven by several factors. Firstly, we aimed to consider approaches that could be feasibly adapted for GNN test prioritization. Secondly, the chosen techniques have been demonstrated as effective for DNNs in the existing literature [10], [55], [56]. Lastly, open-source implementations of these techniques are available.

- **DeepGini** [10] employs the Gini coefficient as a statistical measure of the likelihood of misclassification, thereby enabling the ranking of test inputs. The calculation of the Gini score is presented in Formula 8.

$$\xi(x) = 1 - \sum_{i=1}^{N} (p_i(x))^2 \tag{8}$$

  where $\xi(x)$ refers to the likelihood of the test input $x$ being misclassified. $p_i(x)$ refers to the probability that the test input $x$ is predicted to be label $i$. $N$ refers to the number of labels.
- **Vanilla Softmax** [55] calculates the difference between the value of 1 and the maximum activation probability in the output softmax layer. Formula 9 clearly depicts the calculation process.

$$\mathrm{V}(x) = 1 - \max_{c=1}^{C} l_c(x) \tag{9}$$

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

- **Prediction-Confidence Score (PCS)** PCS [55] measures the difference between the predicted class and the second most confident class in softmax likelihood. PCS is calculated by Formula 10. Low PCS values indicate high probability of being misclassified.
$$P(x) = l_k(x) - l_j(x) \tag{10}$$
where $l_k(x)$ refers to the most confident prediction probability. $l_j(x)$ refers to the second most confident prediction probability.
- **Entropy** Entropy [55] measures uncertainty in a classification model's prediction for a given test by computing the entropy of the softmax likelihood.
- **GraphPrior** GraphPrior [17] is a test prioritization method specifically designed for GNNs. GraphPrior generates mutated models for GNNs and regards tests that kill many mutated models as more likely to be misclassified.
- **Random selection** [57] In random selection, the order of execution for test inputs is determined randomly.

## D. GNN Subjects

*1) Graph Datasets:* Our study utilizes four benchmark datasets commonly used in the field of graph neural networks (GNNs). The Cora and CiteSeer datasets are composed of machine learning publications, represented as nodes in a graph structure, with edges representing citation links between the publications. The PubMed dataset, on the other hand, contains bio-medicine publications. The LastFM Asia Social Network dataset, consists of the relationships between users on the Last.fm music service in Asia, where users are represented as nodes and their mutual follower relationships are represented as edges. These datasets have been widely adopted in existing research on graph neural networks [58], [59], [60], [61], [62].

Overall, we built 124 subjects to evaluate the effectiveness of NodeRank, including 16 subjects of natural datasets and 108 subjects of adversarial datasets.

- **Cora** [63] Cora comprises 2,708 scientific publications and 5,429 links between them. Publications are considered nodes and are classified into seven classes.
- **CiteSeer** [63] CiteSeer is composed of 3,327 scientific publications and 4,732 links between them. Publications (nodes) are classified into six classes.
- **PubMed** [63] PubMed is composed of 19,717 diabetes-related publications and 44,338 links between them. Publications (nodes) are classified into three classes.
- **LastFM Asia Social Network** [64] LastFM Asia Social Network comprises 7,624 nodes and 27,806 edges.

*2) GNN Models:* We consider four GNN models which have been widely studied in the literature of neural network testing, specifically under adversarial attacks.

- **Graph Convolutional Network (GCN)** [41] is a class of neural networks that use graph convolutions. GCN leverages the information of edges to aggregate node information to generate new node representations.
- **Graph Attention Network (GAT)** [42] introduces a graph attention layer to weigh the importance of different nodes within a neighborhood. Each node is assigned an attention score so that more important neighbors can be identified.
- **Topology Adaptive GCN (TAGCN)** [44] designs a set of fixed-size learnable filters to perform convolution operations on graphs. These filters adapt to the topology of the graph while it is scanned for convolution.
- **Graph Sample and Aggregate (GraphSAGE)** [43] generates node embeddings through sampling and aggregating features of neighbor nodes. For computational efficiency, GraphSAGE samples a fixed number of neighbors for each node.

## E. Graph Adversarial Attacks

In RQ2, we aim to investigate the effectiveness of NodeRank on test inputs generated through diverse graph adversarial attacks. Graph adversarial attacks refer to the manipulation of the graph structure or node features to generate graph adversarial perturbations that fool the GNN models. To evaluate the performance of NodeRank against such attacks, we applied a range of adversarial attacks in our experiments. We introduced these attacks as follows.

- **Delete internally, connect externally (DICE)** [31] DICE randomly inserts or deletes an edge for each perturbation. DICE follows two crucial rules: 1) only removing edges between nodes that are from the same class, and 2) only inserting nodes that are from different classes.
- **Min-max attack (MMA)** [32] The min-max attack is a type of untargeted white-box GNN attack, which formulates the attack problem as a min-max optimization problem. In this setup, the inner maximization objective is to update the model's parameters ($\theta$) by maximizing the attack loss, and it can be efficiently solved using gradient ascent. Meanwhile, the outer minimization is achieved using the Projected Gradient Descent (PGD) [65] algorithm, which iteratively perturbs the graph within a bounded $\ell_p$ norm constraint to ensure that the generated perturbations are not too large.
- **Node embedding attack-Add (NEAA)** [33] In the node embedding attack-add, attackers have the ability to manipulate the original graph structure by adding new edges while ensuring that a predetermined budget constraint is not exceeded.
- **Node embedding attack-Remove (NEAR)** [33] In the Node embedding attack-Remove, adversarial attacks are aimed at modifying the original graph structure by selectively removing edges while adhering to a budget constraint.
- **PGD attack (PGD)** [32] The PGD attack leverages the Projected Gradient Descent (PGD) algorithm to search for optimal structural perturbations to attack GNNs.
- **Random Attack-Add (RAA)** [34] RAA randomly adds edges to the input graph to generate perturbations.
- **Random Attack-Remove (RAR)** [34] RAR randomly removes edges to the input graph to generate perturbations.
- **Random Attack-Flip (RAF)** [34] RAF randomly flips edges to the input graph to generate perturbations.

### *F. Variants of NodeRank*

In this paper, when using NodeRank, we refer to the approach that utilizes the Sum-based ensemble learning method (cf. Section III-D) on top of the four considered base models, namely Logistic Regression [46], Random Forest [47], XGBoost [48], and LightGBM [49]. We also implemented two variants using the stacking-based, and voting-based ensemble methods.

*1) NodeRank$^S$:* With this variant, we implemented a stacking-based ensemble method, which uses meta-learning [66] to learn from the outputs of base ranking models to make more accurate predictions. Given a GNN model $M$ that classified nodes into $n$ classes and a test set $T_{test}$, NodeRank$^S$ performs as follows: (1) first, each base ranking model $RM_i$ is trained using mutation features of the training input set $T_{train}$ of $M$; (2) then, NodeRank uses the output of each ranking model to create a new dataset. More specifically, NodeRank$^S$ inputs the mutation results of the training set to each ranking model to obtain the outputs. For each training input, NodeRank$^S$ obtains four probability scores, which will be considered as new features, while the label is 1 or 0. Here, 1 means the training input is misclassified by the GNN model $M$, while 0 means the training input is correctly classified. Since the training set has ground truth for each input, in this way, we build a new dataset. (3) NodeRank$^S$ uses the new dataset to train the meta-learner. Here, each input has four features, which are the outputs from the four ranking models. The ground truth is whether an input is misclassified by the GNN model $M$. (4) After training the meta-learner, NodeRank$^S$ inputs the mutation results of the test set $T_{test}$ to ranking models. Then, NodeRank$^S$ inputs the outputs of ranking models to the meta-learner, which will provide a score for each test input in $T_{test}$. Based on the scores, NodeRank$^S$ prioritizes all the test inputs.

*2) NodeRank$^V$:* With this variant, we implemented the majority voting-based ensemble learning method [67] to combine the prediction results of different ranking models. Majority voting sums the predictions for each class and returns the class with the majority vote as the ensemble prediction. Given a GNN model $M$ and a test set $T_{test}$, NodeRank$^V$ performs as follows: (1) first, each base ranking model $RM_i$ is trained using mutation features of the training input set $T_{train}$ of $M$; (2) For a test input in $T_{test}$, NodeRank$^V$ inputs its mutation features to $N$ ranking models, obtaining $N$ scores (i.e., misclassification probabilities) for this input, denoted as $\{S_1, S_2, \cdots, S_N\}$. Then, NodeRank$^S$ transforms each score into 0 or 1. Scores below 0.5 are converted to 0, otherwise to 1. In this way, NodeRank$^S$ obtains an N-length vector for each input. For example, $\{0, 1, \cdots, 0\}$. NodeRank$^S$ regards 1 voting for misclassification (i.e., the input will be misclassified by the GNN model $M$) and 0 voting for correct classification. (3) After voting, for each input, NodeRank$^V$ sums its votes from all ranking models. NodeRank$^V$ ranks all the test inputs based on their votes for misclassification.

### *G. Implementation and Configuration*

We implemented NodeRank in Python based on the PyTorch [68] framework. We also integrate the available implementations of the compared approaches [10], [16], [69] into our experimental pipeline to adapt to the GNN prioritization problem. Regarding the GNN models selected as subjects in our study, the range of their accuracy is: GAT: 71%$\sim$77%, GCN: 70%$\sim$73%, GraphSAGE: 71%$\sim$73%, TAGCN: 72%$\sim$81%. Regarding our mutation rules, for the GNN model mutation, we generated 144 mutants on average. For graph structure mutation, we generated 265 mutants on average. For node feature mutation, we generated 147 mutants on average. Concerning the configurations of node mutation rules in the experiments of this paper, we made the following design choice: We slightly modify attributes, with an offset between 0.005 to 0.015.

We conducted all learning experiments on a high-performance computer cluster, where each cluster node runs a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. For the data processing, we conducted our experiments on a MacBook Pro laptop with Mac OS Big Sur 11.6, Intel Core i9 CPU, and 64 GB RAM. Overall, our experiments involved 124 subjects, of which 16 subjects were based on natural inputs and 108 subjects were based on adversarial inputs.

## V. Experimental Results

For each research question, we present the experimental objective, design, and results before discussing the findings.

### *A. RQ1: Performance of NodeRank*

**Objective:** We evaluate the performance of NodeRank in prioritizing test inputs for GNNs. To that end, we also compare NodeRank against five uncertainty-based test prioritization approaches.

**Experimental design:** We use our initial subjects (4 datasets and 4 GNN models, leading to 16 combinations of *natural inputs*, i.e., without any adversarial attacks introduced). Moreover, we compare NodeRank with 6 test prioritization approaches, which include 1 test prioritization method for GNNs (GraphPrior), 4 test prioritization methods for traditional DNNs (i.e., DeepGini, VanillaSM, PCS, and Entropy), and a baseline method (random selection). Specific details about these compared methods can be found in Section IV-C. All subjects are applied to NodeRank, as well as the six compared approaches. Beyond effectiveness, we also investigated the efficiency of NodeRank by analyzing the time cost of each step involved in its execution. Furthermore, due to the randomness in the GNN model training process, we conducted a statistical analysis to ensure the stability of our findings. Following the prior work [70], we repeated all the experiments 30 times. The following results are the averages obtained from the 30 repeated experiments.

To demonstrate the statistical significance of the improvement of NodeRank relative to the compared test prioritization approaches, we utilized the Mann-Whitney U test [71] to compute the $p$-value of the repeated experimental results. The Mann-Whitney U test is a statistical method used to determine whether there is a notable distinction between two sets

TABLE I
EFFECTIVENESS COMPARISON AMONG NODERANK, RANDOM, DEEPGINI, VANILLASM, PCS, AND ENTROPY IN TERMS OF THE APFD VALUES ON NATURAL DATASETS

| Approach | CiteSeer | | | | Cora | | | | LastFM | | | | PubMed | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN |
| Random | 0.4784 | 0.4893 | 0.4837 | 0.4882 | 0.4912 | 0.5268 | 0.4857 | 0.4782 | 0.4870 | 0.4981 | 0.5142 | 0.5049 | 0.4890 | 0.4972 | 0.5032 | 0.5123 |
| DeepGini | 0.6072 | 0.6197 | 0.6732 | 0.6260 | 0.7080 | 0.7037 | 0.7070 | 0.7650 | 0.5887 | 0.6991 | 0.7820 | 0.7368 | 0.6371 | 0.6995 | 0.6952 | 0.6140 |
| VanillaSM | 0.6519 | 0.6611 | 0.6831 | 0.6534 | 0.7325 | 0.7292 | 0.7283 | 0.7688 | 0.6696 | 0.7437 | 0.7850 | 0.7677 | 0.6630 | 0.7196 | 0.6981 | 0.6583 |
| PCS | 0.6528 | 0.6848 | 0.6767 | 0.6541 | 0.7132 | 0.7239 | 0.7303 | 0.7330 | 0.6925 | 0.7454 | 0.7463 | 0.7548 | 0.6569 | 0.6738 | 0.6660 | 0.6658 |
| Entropy | 0.6045 | 0.6181 | 0.6727 | 0.6165 | 0.7019 | 0.7007 | 0.7025 | 0.7564 | 0.5228 | 0.6411 | 0.7082 | 0.6011 | 0.6402 | 0.7004 | 0.6968 | 0.6155 |
| GraphPrior | 0.6754 | 0.6942 | 0.7103 | 0.6961 | 0.7853 | 0.7883 | 0.7651 | 0.7815 | 0.7746 | 0.7834 | 0.7914 | 0.7792 | 0.7546 | 0.7426 | 0.7534 | 0.7285 |
| **NodeRank** | 0.7319 | 0.7203 | 0.7325 | 0.7199 | 0.8326 | 0.8021 | 0.8121 | 0.8164 | 0.8146 | 0.8151 | 0.8063 | 0.8225 | 0.7714 | 0.7670 | 0.7895 | 0.7795 |

Note: The gray shade indicates the approach with the highest effectiveness.

of data distributions. The Mann-Whitney U test does not require the assumption of normal distribution for the data. Therefore, it can be used for both normal and non-normal distributed data. The Mann-Whitney U test transforms the data into ranks, calculates a test statistic based on these ranks, and uses this as a basis for computing the $p$-value to assess if there is a statistically significant difference between the two sets of data. A $p$-value $<$ 0.05 is generally considered indicative of significance.

Furthermore, in addition to showcasing the average experimental results, we also evaluate the variability of these results in order to ensure a more fair comparison between the effectiveness of NodeRank and existing test prioritization approaches. The specific steps of these experiments are elucidated below:

- **Effectiveness distributions between NodeRank and the compared approaches** As previously mentioned, we conducted 30 repetitions of all experiments. Subsequently, based on the results generated from these 30 repetitions, we used box plots to illustrate the distribution of results for various test prioritization methods. The rationale behind employing box plots is that: 1) they offer an intuitive representation of data distribution, including key statistics like the median, quartiles, and identification of outliers. This visual format enables a quick understanding of data characteristics; 2) Box plots offer a visual tool for easily comparing the distribution of experimental results across various test prioritization approaches. When multiple box plots are displayed side by side, the differences between them can be clearly exhibited.
- **Confidence interval between NodeRank and the compared approaches** Based on the results of 30 repeated experiments, we calculated the confidence interval of each test prioritization approach. Following the existing study [72], we employed Formula 11 to compute the upper and lower bounds of the confidence interval. We calculated the confidence intervals for different test prioritization methods across two metrics (PFD and APFD) and two scenarios (natural and adversarial datasets).

$$\left(\bar{X} - Z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}, \bar{X} + Z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}\right) \tag{11}$$

where $\bar{X}$ represents the average value, $\sigma$ represents the standard deviation, $n$ represents the sample size, and $Z_{\alpha/2}$ represents the confidence coefficient.

**Results:** The experimental results of RQ1 are presented in Tables I–VII, Figs. 2 and 3. We highlight the approach with the highest effectiveness in grey to facilitate quick and easy interpretation of the results. Table I presents the APFD scores of NodeRank and the compared approaches on each subject (i.e., a combination of a natural dataset and GNN model). We see that NodeRank consistently outperforms all compared approaches on all 16 subjects (i.e., 16 Best cases for NodeRank). Moreover, the APFD range for NodeRank is 0.7199 to 0.8326, while GraphPrior (the test prioritization method specifically designed for GNNs) falls within the range of 0.6754 to 0.7883. Additionally, the APFD range for other test prioritization methods varies from 0.4784 to 0.7850. Table II presents an in-depth assessment of NodeRank's effectiveness in comparison to other approaches, including the number of best cases achieved by each approach, the average APFD, and the improvement that NodeRank offers over the compared methods. We see that the average APFD for NodeRank is 0.7883, while the average APFD for GraphPrior is 0.7502. In contrast, the average APFD range for other test prioritization methods falls between 0.4954 and 0.7071. When compared to GraphPrior, NodeRank exhibits an average improvement of 4.41%, while its improvement relative to other comparative methods ranges from 10.78% to 58.11%.

TABLE II
PERFORMANCE IMPROVEMENT OF NODERANK ON THE 16 INITIAL SUBJECTS (I.E., 4 NATURAL INPUT SETS ON 4 GNN MODELS)

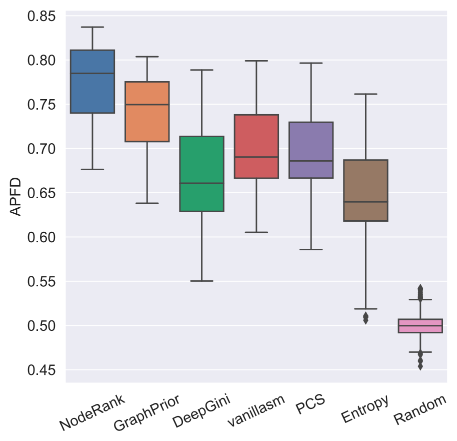
| Approach | # Best cases | Average APFD | Improvement(%) |
|---|---|---|---|
| Random | 0 | 0.4954 | 58.11 |
| DeepGini | 0 | 0.6788 | 15.39 |
| VanillaSM | 0 | 0.7071 | 10.78 |
| PCS | 0 | 0.6981 | 12.20 |
| Entropy | 0 | 0.6513 | 20.27 |
| GraphPrior | 0 | 0.7502 | 4.41 |
| **NodeRank** | 16 | 0.7833 | - |

We present further evidence of the high effectiveness of NodeRank in the context of test prioritization by utilizing the PFD (Percentage of Fault Detected) metric. The corresponding experimental results are presented in Table III. Our analysis demonstrates that NodeRank consistently surpasses GraphPrior, all the confidence-based approaches, and random selection in terms of average PFD, regardless of the proportion of prioritized tests. Furthermore, the effectiveness of NodeRank is visually apparent in Fig. 3. In the figure, NodeRank is represented by the red line, GraphPrior by the blue line, and the baseline method by the pink line. It is evident that NodeRank consistently outperforms GraphPrior,

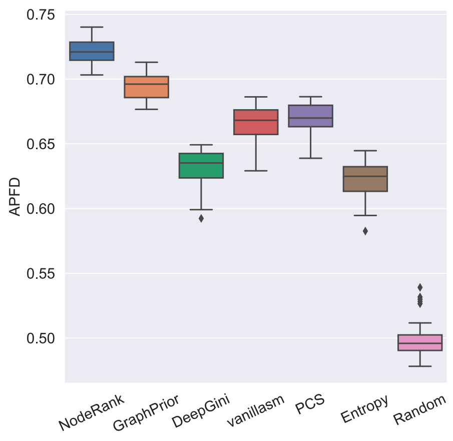
TABLE III
AVERAGE COMPARISON RESULTS AMONG NODERANK AND THE COMPARED APPROACHES IN TERMS OF PFD

| Data | Approach | PFD-10 | PFD-20 | PFD-30 | PFD-40 | PFD-50 | PFD-60 |
|---|---|---|---|---|---|---|---|
| CiteSeer | Random | 0.0859 | 0.1758 | 0.2802 | 0.3859 | 0.4776 | 0.5753 |
| | DeepGini | 0.1999 | 0.3393 | 0.4609 | 0.5796 | 0.6849 | 0.7705 |
| | VanillaSM | 0.2104 | 0.3805 | 0.5254 | 0.6471 | 0.7253 | 0.8077 |
| | PCS | 0.2103 | 0.3858 | 0.5267 | 0.6478 | 0.7513 | 0.8217 |
| | Entropy | 0.1991 | 0.3402 | 0.4542 | 0.5772 | 0.6766 | 0.7683 |
| | GraphPrior | 0.2355 | 0.4431 | 0.6024 | 0.7253 | 0.7827 | 0.8235 |
| | **NodeRank** | 0.2684 | 0.5078 | 0.6790 | 0.7752 | 0.8267 | 0.8697 |
| Cora | Random | 0.0938 | 0.1955 | 0.2996 | 0.3893 | 0.4971 | 0.5822 |
| | DeepGini | 0.2555 | 0.4647 | 0.6193 | 0.7406 | 0.8293 | 0.8831 |
| | VanillaSM | 0.2718 | 0.4906 | 0.6461 | 0.7676 | 0.8541 | 0.9170 |
| | PCS | 0.2406 | 0.4386 | 0.6061 | 0.7513 | 0.8422 | 0.9103 |
| | Entropy | 0.2551 | 0.4627 | 0.6100 | 0.7325 | 0.8250 | 0.8759 |
| | GraphPrior | 0.3025 | 0.6021 | 0.7254 | 0.8013 | 0.8923 | 0.9215 |
| | **NodeRank** | 0.3434 | 0.6764 | 0.8682 | 0.9113 | 0.9342 | 0.9468 |
| LastFM | Random | 0.1021 | 0.1983 | 0.3041 | 0.4045 | 0.5039 | 0.5994 |
| | DeepGini | 0.2476 | 0.4549 | 0.6042 | 0.7128 | 0.7898 | 0.8548 |
| | VanillaSM | 0.2560 | 0.4939 | 0.6606 | 0.7814 | 0.8658 | 0.9177 |
| | PCS | 0.2253 | 0.4593 | 0.6527 | 0.7883 | 0.8698 | 0.9143 |
| | Entropy | 0.2472 | 0.4264 | 0.5190 | 0.6022 | 0.6705 | 0.7214 |
| | GraphPrior | 0.3015 | 0.5324 | 0.7612 | 0.8563 | 0.8746 | 0.9237 |
| | **NodeRank** | 0.3487 | 0.6833 | 0.8621 | 0.9083 | 0.9297 | 0.9473 |
| PubMed | Random | 0.1015 | 0.2023 | 0.3027 | 0.3989 | 0.4959 | 0.5957 |
| | DeepGini | 0.2344 | 0.4026 | 0.5463 | 0.6407 | 0.7226 | 0.7959 |
| | VanillaSM | 0.2270 | 0.4034 | 0.5649 | 0.6935 | 0.7851 | 0.8516 |
| | PCS | 0.1968 | 0.3811 | 0.5422 | 0.6640 | 0.7630 | 0.8313 |
| | Entropy | 0.2348 | 0.4028 | 0.5467 | 0.6424 | 0.7264 | 0.7994 |
| | GraphPrior | 0.3021 | 0.5163 | 0.6582 | 0.7535 | 0.8192 | 0.8746 |
| | **NodeRank** | 0.3463 | 0.6258 | 0.7744 | 0.8359 | 0.8748 | 0.9062 |

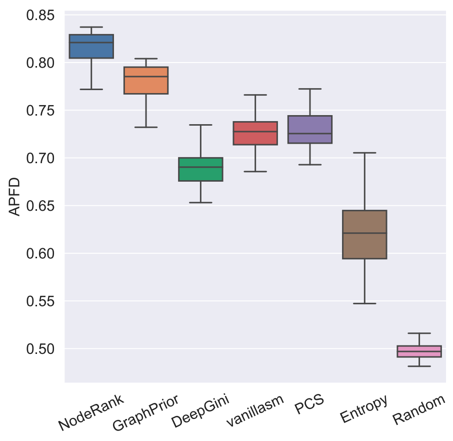
Note: The gray shade indicates the approach with the highest effectiveness.



(a) All natural subjects



(b) CiteSeer, TAGCN



(c) LastFM, GCN

Fig. 2. Effectiveness distributions between NodeRank and the compared approaches on natural test inputs.

TABLE IV
CONFIDENCE INTERVAL OF NODERANK AND THE COMPARED APPROACHES IN TERMS OF APFD ON NATURAL TEST INPUTS

| Approach | Lower Bound | Upper Bound |
|---|---|---|
| Random | 0.4942 | 0.4966 |
| DeepGini | 0.6743 | 0.6832 |
| VanillaSM | 0.7032 | 0.7109 |
| PCS | 0.6945 | 0.7016 |
| Entropy | 0.6467 | 0.6558 |
| GraphPrior | 0.7467 | 0.7536 |
| **NodeRank** | 0.7798 | 0.7867 |

Note: The gray shade indicates the approach with the highest effectiveness.

all the confidence-based approaches, and the baseline. These experimental findings further confirm the high effectiveness of NodeRank.

Table VI presents the results of statistical analysis. We use the Mann–Whitney U test [71] as the metric to calculate the p-value of the experimental results. Our objective is to demonstrate that the improvement of NodeRank over other testing methods is statistically significant. Within Table VI, we see that the range of p-values is from $7.7245 \times 10^{-7}$ to 0.0092. These values are all less than 0.05, indicating that the improvement of NodeRank compared to other test prioritization methods is statistically significant.

TABLE V
CONFIDENCE INTERVAL OF NODERANK AND THE COMPARED APPROACHES IN TERMS OF PFD ON NATURAL TEST INPUTS

| Approach | PFD-10 | | PFD-20 | | PFD-30 | | PFD-40 | | PFD-50 | | PFD-60 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper |
| Random | 0.0917 | 0.1008 | 0.1887 | 0.1989 | 0.2907 | 0.3024 | 0.3879 | 0.4007 | 0.4872 | 0.4994 | 0.5817 | 0.5933 |
| DeepGini | 0.2280 | 0.2411 | 0.4101 | 0.4197 | 0.5523 | 0.5628 | 0.6620 | 0.6743 | 0.7496 | 0.7609 | 0.8197 | 0.8305 |
| VanillaSM | 0.2346 | 0.2482 | 0.4373 | 0.4477 | 0.5928 | 0.6032 | 0.7175 | 0.7265 | 0.8016 | 0.8129 | 0.8675 | 0.8791 |
| PCS | 0.2124 | 0.2231 | 0.4112 | 0.4222 | 0.5760 | 0.5877 | 0.7071 | 0.7183 | 0.7998 | 0.8111 | 0.8637 | 0.8737 |
| Entropy | 0.2285 | 0.2398 | 0.4012 | 0.4128 | 0.5282 | 0.5378 | 0.6319 | 0.6426 | 0.7186 | 0.7309 | 0.7870 | 0.7953 |
| GraphPrior | 0.2811 | 0.2920 | 0.5175 | 0.5291 | 0.6826 | 0.6912 | 0.7773 | 0.7899 | 0.8359 | 0.8477 | 0.8814 | 0.8923 |
| **NodeRank** | 0.3215 | 0.3322 | 0.6167 | 0.6294 | 0.7910 | 0.8014 | 0.8534 | 0.8631 | 0.8865 | 0.8967 | 0.9118 | 0.9218 |

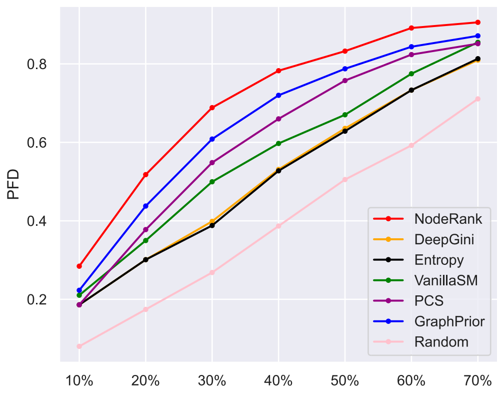
Note: The gray shade indicates the approach with the highest effectiveness.

TABLE VI
STATISTICAL ANALYSIS ON NATURAL TEST INPUTS (IN TERMS OF P-VALUE UNDER THE MANN–WHITNEY U TEST)

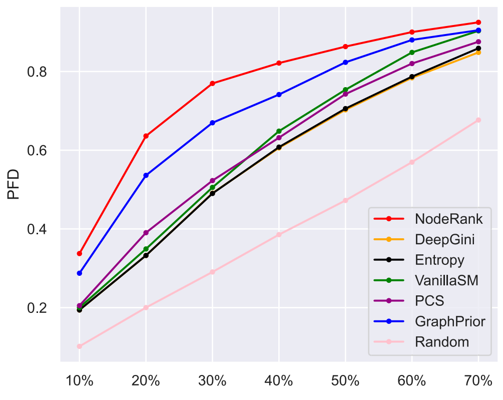
| Mann–Whitney U test | NodeRank vs Random | NodeRank vs DeepGini | NodeRank vs VanillaSM | NodeRank vs PCS | NodeRank vs Entropy | NodeRank vs GraphPrior |
|---|---|---|---|---|---|---|
| p-value | $7.7245 \times 10^{-7}$ | $1.1175 \times 10^{-5}$ | $9.5154 \times 10^{-5}$ | $2.5438 \times 10^{-5}$ | $1.6231 \times 10^{-6}$ | 0.0092 |

TABLE VII
TIME COST OF NODERANK AND THE COMPARED APPROACHES

| Time cost | Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | **NodeRank** | Random | GraphPrior | DeepGini | VanillaSM | PCS | Entropy |
| Mutant generation | 35 min | - | 35 min | - | - | - | - |
| Feature extraction | 30 s | - | 20 s | - | - | - | - |
| Ranking model training | 3 min | - | 3 min | - | - | - | - |
| Prediction | <1 s | <1 s | <1 s | <1 s | <1 s | <1 s | <1 s |



(a) CiteSeer, TAGCN



(b) PubMed, GAT

Fig. 3. Test prioritization effectiveness among NodeRank and the compared approaches for CiteSeer with TAGCN and PubMed with GAT. X-Axis: the percentage of prioritized tests; Y-Axis: the percentage of detected misclassified tests.

Moreover, Fig. 2 presents and compares the effectiveness (in terms of APFD) of NodeRank with other test prioritization methods using box plots. The box plots highlight the distribution of results of multiple repeated experiments for NodeRank and other test prioritization methods. In Fig. 2, we see that, in terms of the median, the median APFD value of NodeRank exceeds that of other test prioritization methods across all natural datasets. Moreover, in the presented two specific examples shown in the box plots, which respectively correspond to subject CiteSeer, TAGCN, and subject LastFM, GCN, we can also see that the median of NodeRank from repeated experiments is the highest.

Regarding the quartile range, NodeRank's quartile range (i.e., the height of the box) exhibits some variations across different datasets, but overall, its upper quartile is higher than that of other methods. Analyzing outliers, the box plots do not show significant outliers, indicating that NodeRank's performance across different datasets is relatively stable, with no extreme cases of inefficiency. In summary, we conclude that NodeRank outperforms all compared testing prioritization methods in terms of APFD based on the distribution of data from multiple experimental results. This demonstrates that NodeRank exhibits better effectiveness in test prioritization compared to other methods.

Furthermore, we calculated the confidence intervals for all test prioritization methods, and the experimental results are presented in Tables IV and V. In Table IV, we see that NodeRank's APFD has the highest lower and upper bounds compared to other test prioritization methods, with values of 0.7798 and 0.7867, respectively. Notably, NodeRank's lower bound (0.7798) even exceeds the upper bounds of all other comparative methods. GraphPrior's upper bound is 0.7536, while the upper bounds for other test prioritization methods range from 0.4966 to 0.7109. Table V exhibits the confidence intervals of all test prioritization methods in terms of PFD. The gray highlights indicate the test prioritization approaches that achieve the maximum PFD in this scenario. In Table V, we see that NodeRank also demonstrates the highest lower and upper bounds in terms of PFD compared to other test prioritization methods when prioritizing different ratios of tests. These experimental findings highlight that, in terms of confidence intervals, NodeRank's effectiveness exceeds that of the comparative test prioritization methods.

In addition to its effectiveness, we also present an analysis of NodeRank's efficiency in Table VII. We offer a comprehensive breakdown of the time taken by each step in NodeRank and compare it with GraphPrior, the confidence-based test prioritization methods, and the baseline approach (random selection). As shown in Table VII, the time required for NodeRank is divided into four parts: mutant generation, feature extraction, ranking model training, and NodeRank prediction. Among these steps, mutant generation is found to be the most time-consuming, taking approximately 35 minutes, followed by ranking model training, which takes approximately 3 minutes. Overall, NodeRank requires a total time of approximately 38 minutes. However, it is worth noting that the prediction time of NodeRank is extremely fast, taking less than 1s once the ranking model is trained, and the mutation features are extracted. The overall runtime of GraphPrior is similar to NodeRank, approximately 38 minutes. In contrast, the confidence-based test prioritization methods have an overall runtime of less than 1 second. While NodeRank is less efficient than the uncertainty-based test prioritization approaches (which takes less than 1s), its time cost remains acceptable compared to the prohibitively expensive manual labeling.

**Answer to RQ1:** On natural test inputs, NodeRank consistently exhibits better effectiveness compared to GraphPrior, all confidence-based approaches, and the baseline method across all subjects, as evident from both APFD and PFD metrics.

In terms of APFD, NodeRank showcases an average improvement of 4.41% and 58.11% over the compared approaches. Additionally, the efficiency of NodeRank is within an acceptable range, thereby demonstrating its practical usefulness.

## *B. RQ2: Prioritization of Adversarial Inputs*

**Objective:** We evaluate the effectiveness of NodeRank on adversarial test inputs. We assume that natural test inputs (cf. RQ1) can easily discriminate which ones are more likely to reveal bugs. In contrast, with adversarial inputs, by construction, they are all generated to make the probability of the wrong classification label as high as possible. Thus, a test input prioritization on adversarial inputs may be challenged in ranking them adequately. Yet, such prioritization is still necessary to ensure a fast assessment of GNN model robustness.

**Experimental design:** To investigate the effectiveness of NodeRank on adversarial datasets, we generated adversarial test inputs using eight graph adversarial attack methods [31], [33], [34]. We set the attack level to 0.3, which indicates that 30% of the test inputs in the test set are adversarial tests. It is worth noting that a high attack level, such as 90%, would result in a significant proportion of adversarial test inputs. Under such circumstances, any prioritization method could potentially select a larger number of bug cases, making it difficult to effectively demonstrate the efficacy of NodeRank. Thus, to ensure a proper evaluation of NodeRank and the compared approaches, we selected a reasonable attack level (i.e., 0.3), which effectively limits the proportion of adversarial test inputs.

Eventually, we construct 108 subjects (i.e., a combination of a GNN model and an adversarial inputs set). Consistent with the experimental design employed in RQ1, we evaluate the prioritization effectiveness of NodeRank and the compared approaches using both the APFD and PFD metrics. Similar to RQ1, we conducted 30 repetitions of all experiments and reported the average outcomes. Aside from presenting the average experimental findings, we assessed the variability of these results to ensure a fairer comparison between the effectiveness of NodeRank and existing test prioritization methods. Detailed steps for these experiments can be found in the experimental design of RQ1 (refer to Section V-A).

**Results:** The experimental results of RQ2 are presented in Tables VIII–XIV, Figs. 4 and 5. Table VIII presents the APFD scores of NodeRank and the compared approaches on DICE-based graph adversarial inputs. Again, NodeRank performs the best across all subjects. Experiment results on all the subjects are available on our GitHub.[2]

Table IX presents the average APFD values for NodeRank and the compared approaches, as well as the average improvement of NodeRank over the compared approaches across different adversarial attacks. We can see that, across all cases, NodeRank consistently outperforms GraphPrior, confidence-based approaches, and random selection. Specifically, NodeRank achieves an average APFD ranging from 0.7570 to 0.8041, whereas GraphPrior averages between 0.7046 and 0.7625. The remaining testing prioritization methods show APFD ranges from 0.4922 to 0.7337. In terms of improvement over GraphPrior, NodeRank demonstrates an average improvement ranging from 4.69% to 8.78%. NodeRank's improvement over the other testing prioritization methods varies from 6.72% to 62.15%.

[2]https://github.com/yinghuali/NodeRank/tree/main/results

TABLE VIII
TEST PRIORITIZATION PERFORMANCE (APFD SCORES) ON DICE-BASED GRAPH ADVERSARIAL TEST INPUTS

| Approach | CiteSeer | | | | Cora | | | | LastFM | | | | PubMed | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN |
| Random | 0.4867 | 0.4986 | 0.4937 | 0.5153 | 0.4938 | 0.5250 | 0.4817 | 0.5100 | 0.5031 | 0.4907 | 0.5004 | 0.4973 | 0.4997 | 0.4953 | 0.4929 | 0.4940 |
| DeepGini | 0.5893 | 0.6059 | 0.6550 | 0.6168 | 0.6878 | 0.6873 | 0.7061 | 0.7269 | 0.5883 | 0.6897 | 0.7685 | 0.7136 | 0.6363 | 0.6726 | 0.6846 | 0.6145 |
| VanillaSM | 0.6159 | 0.6418 | 0.6695 | 0.6335 | 0.7058 | 0.7113 | 0.7189 | 0.7359 | 0.6559 | 0.7294 | 0.7720 | 0.7502 | 0.6567 | 0.6875 | 0.6875 | 0.6488 |
| PCS | 0.6058 | 0.6494 | 0.6639 | 0.6304 | 0.6803 | 0.6974 | 0.7065 | 0.7066 | 0.6696 | 0.7289 | 0.7342 | 0.7428 | 0.6453 | 0.6379 | 0.6538 | 0.6509 |
| Entropy | 0.5874 | 0.6044 | 0.6536 | 0.6123 | 0.6827 | 0.6839 | 0.7025 | 0.7175 | 0.5364 | 0.6478 | 0.6961 | 0.5828 | 0.6384 | 0.6731 | 0.6860 | 0.6157 |
| GraphPrior | 0.7013 | 0.6955 | 0.7143 | 0.6745 | 0.7525 | 0.7436 | 0.7515 | 0.7537 | 0.7652 | 0.7561 | 0.7827 | 0.7732 | 0.7067 | 0.7037 | 0.7051 | 0.7038 |
| **NodeRank** | 0.7249 | 0.7128 | 0.7402 | 0.7171 | 0.8054 | 0.7963 | 0.8031 | 0.7859 | 0.8034 | 0.8059 | 0.8054 | 0.8040 | 0.7543 | 0.7465 | 0.7752 | 0.7699 |

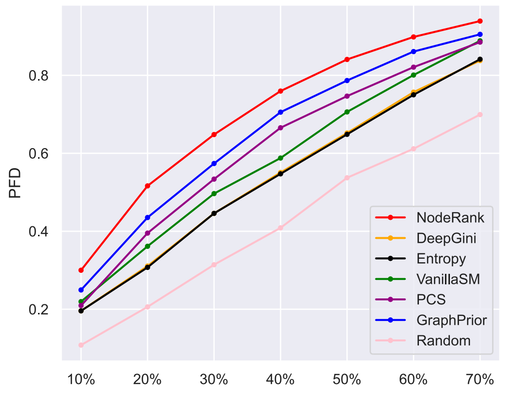
Note: The gray shade indicates the approach with the highest effectiveness.

TABLE IX
OVERALL COMPARISON RESULTS ON GRAPH ADVERSARIAL DATASETS

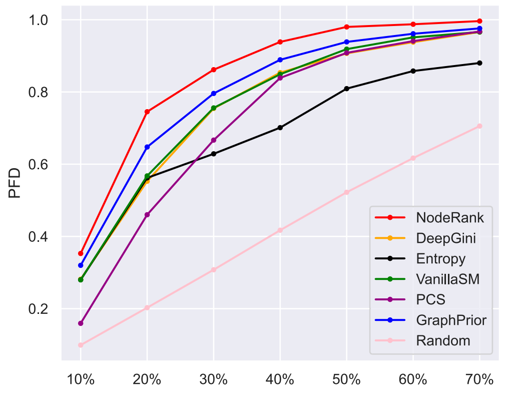
| Attack | Approach | Average performance score (APFD) | | | | Improvement(of APFD) of NodeRank over the compared approaches | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | GAT | GCN | GraphSAGE | TAGCN | GAT | GCN | GraphSAGE | TAGCN |
| DICE | Random | 0.4958 | 0.5024 | 0.4922 | 0.5042 | 55.71% | 52.35% | 58.68% | 52.56% |
| | DeepGini | 0.6254 | 0.6639 | 0.7036 | 0.6679 | 23.44% | 15.29% | 11.03% | 15.17% |
| | VanillaSM | 0.6586 | 0.6925 | 0.7120 | 0.6921 | 17.22% | 10.53% | 9.69% | 11.14% |
| | PCS | 0.6503 | 0.6784 | 0.6896 | 0.6827 | 18.71% | 12.82% | 13.25% | 12.67% |
| | Entropy | 0.6112 | 0.6523 | 0.6846 | 0.6321 | 26.31% | 17.34% | 14.08% | 21.69% |
| | GraphPrior | 0.7314 | 0.7247 | 0.7384 | 0.7263 | 5.54% | 5.61% | 5.76% | 5.91% |
| | **NodeRank** | 0.7720 | 0.7654 | 0.7810 | 0.7692 | - | - | - | - |
| MMA | Random | 0.5283 | 0.5128 | 0.5161 | 0.4775 | 46.60% | 47.62% | 50.34% | 60.27% |
| | DeepGini | 0.6591 | 0.6616 | 0.6960 | 0.6950 | 17.51% | 14.42% | 11.48% | 10.12% |
| | VanillaSM | 0.6857 | 0.6923 | 0.7093 | 0.7104 | 12.95% | 9.35% | 9.39% | 7.73% |
| | PCS | 0.6662 | 0.6980 | 0.7034 | 0.6952 | 16.26% | 8.45% | 10.31% | 10.08% |
| | Entropy | 0.6551 | 0.6591 | 0.6938 | 0.6881 | 18.23% | 14.85% | 11.83% | 11.22% |
| | GraphPrior | 0.7258 | 0.7126 | 0.7392 | 0.7266 | 6.70% | 6.23% | 4.96% | 5.32% |
| | **NodeRank** | 0.7745 | 0.7570 | 0.7759 | 0.7653 | - | - | - | - |
| NEAA | Random | 0.5079 | 0.5053 | 0.5009 | 0.4938 | 54.54% | 54.98% | 59.19% | 60.33% |
| | DeepGini | 0.6311 | 0.6869 | 0.7238 | 0.6864 | 24.37% | 14.00% | 10.17% | 15.34% |
| | VanillaSM | 0.6686 | 0.7170 | 0.7301 | 0.7172 | 17.39% | 9.22% | 9.23% | 10.39% |
| | PCS | 0.6630 | 0.6931 | 0.7062 | 0.7032 | 18.39% | 12.99% | 12.95% | 12.59% |
| | Entropy | 0.6114 | 0.6708 | 0.7005 | 0.6410 | 28.38% | 16.74% | 13.83% | 23.51% |
| | GraphPrior | 0.7421 | 0.7368 | 0.7521 | 0.7433 | 5.76% | 6.28% | 6.02% | 6.51% |
| | **NodeRank** | 0.7849 | 0.7831 | 0.7974 | 0.7917 | - | - | - | - |
| NEAR | Random | 0.4936 | 0.5016 | 0.4946 | 0.5134 | 61.93% | 58.19% | 62.15% | 56.62% |
| | DeepGini | 0.6426 | 0.6955 | 0.7241 | 0.7005 | 24.39% | 14.09% | 10.76% | 14.79% |
| | VanillaSM | 0.6850 | 0.7237 | 0.7337 | 0.7269 | 16.69% | 9.64% | 9.31% | 10.62% |
| | PCS | 0.6841 | 0.7061 | 0.7096 | 0.7144 | 16.84% | 12.38% | 13.02% | 12.56% |
| | Entropy | 0.6206 | 0.6771 | 0.6976 | 0.6523 | 28.79% | 17.19% | 14.97% | 23.27% |
| | GraphPrior | 0.7352 | 0.7548 | 0.7625 | 0.7581 | 8.71% | 5.12% | 5.18% | 6.06% |
| | **NodeRank** | 0.7993 | 0.7935 | 0.8020 | 0.8041 | - | - | - | - |
| PGD | Random | 0.5043 | 0.5114 | 0.5026 | 0.5090 | 56.81% | 52.07% | 57.02% | 53.14% |
| | DeepGini | 0.6378 | 0.6764 | 0.7239 | 0.7137 | 23.99% | 14.98% | 9.02% | 9.22% |
| | VanillaSM | 0.6839 | 0.7133 | 0.7336 | 0.7304 | 15.63% | 9.03% | 7.58% | 6.72% |
| | PCS | 0.6805 | 0.7187 | 0.7176 | 0.7133 | 16.21% | 8.21% | 9.98% | 9.28% |
| | Entropy | 0.6169 | 0.6593 | 0.6997 | 0.6644 | 28.19% | 17.96% | 12.79% | 17.32% |
| | GraphPrior | 0.7491 | 0.7324 | 0.7403 | 0.7387 | 5.56% | 6.19% | 6.60% | 5.52% |
| | **NodeRank** | 0.7908 | 0.7778 | 0.7892 | 0.7795 | - | - | - | - |
| RAA | Random | 0.4981 | 0.4951 | 0.5027 | 0.5018 | 53.88% | 55.10% | 55.06% | 54.15% |
| | DeepGini | 0.6299 | 0.6660 | 0.7084 | 0.6709 | 21.69% | 15.30% | 10.04% | 15.29% |
| | VanillaSM | 0.6596 | 0.6964 | 0.7160 | 0.6972 | 16.21% | 10.27% | 8.87% | 10.94% |
| | PCS | 0.6459 | 0.6824 | 0.6931 | 0.6836 | 18.67% | 12.53% | 12.47% | 13.15% |
| | Entropy | 0.6166 | 0.6553 | 0.6910 | 0.6360 | 24.31% | 17.18% | 12.81% | 21.62% |
| | GraphPrior | 0.7046 | 0.7261 | 0.7312 | 0.7258 | 8.78% | 5.75% | 6.60% | 6.57% |
| | **NodeRank** | 0.7665 | 0.7679 | 0.7795 | 0.7735 | - | - | - | - |
| RAF | Random | 0.4990 | 0.4964 | 0.5003 | 0.5004 | 54.31% | 55.00% | 56.69% | 54.66% |
| | DeepGini | 0.6199 | 0.6660 | 0.7074 | 0.6724 | 24.21% | 15.53% | 10.81% | 15.10% |
| | VanillaSM | 0.6519 | 0.6971 | 0.7157 | 0.6984 | 18.12% | 10.37% | 9.53% | 10.81% |
| | PCS | 0.6415 | 0.6829 | 0.6937 | 0.6828 | 20.03% | 12.67% | 13.00% | 13.34% |
| | Entropy | 0.6062 | 0.6550 | 0.6882 | 0.6374 | 27.02% | 17.47% | 13.91% | 21.42% |
| | GraphPrior | 0.7109 | 0.7257 | 0.7369 | 0.7281 | 8.34% | 6.02% | 6.37% | 6.29% |
| | **NodeRank** | 0.7702 | 0.7694 | 0.7839 | 0.7739 | - | - | - | - |
| RAR | Random | 0.5134 | 0.5015 | 0.5043 | 0.5024 | 52.84% | 53.84% | 56.61% | 56.35% |
| | DeepGini | 0.6312 | 0.6765 | 0.7063 | 0.6895 | 24.32% | 14.04% | 11.82% | 13.92% |
| | VanillaSM | 0.6723 | 0.7080 | 0.7153 | 0.7115 | 16.72% | 8.97% | 10.42% | 10.40% |
| | PCS | 0.6700 | 0.6990 | 0.6994 | 0.7039 | 17.12% | 10.37% | 12.93% | 11.59% |
| | Entropy | 0.6144 | 0.6620 | 0.6875 | 0.6542 | 27.72% | 16.54% | 14.88% | 20.07% |
| | GraphPrior | 0.7403 | 0.7325 | 0.7421 | 0.7362 | 5.99% | 5.32% | 6.42% | 6.69% |
| | **NodeRank** | 0.7847 | 0.7715 | 0.7898 | 0.7855 | - | - | - | - |

Note: The gray shade indicates the approach with the highest effectiveness.

Table XIII presents the results of statistical analysis on adversarial datasets. The adopted approach (Mann-Whitney U test method) for calculating the p-value is explained in RQ1 (Section V-A). Within Table XIII, we see that the range of p-values is from $2.4541 \times 10^{-8}$ to 0.0057. All these values fall below 0.05, indicating that the improvement of NodeRank

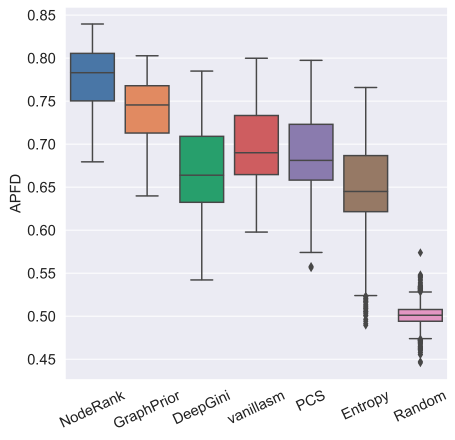


(a) MMA, CiteSeer, GCN

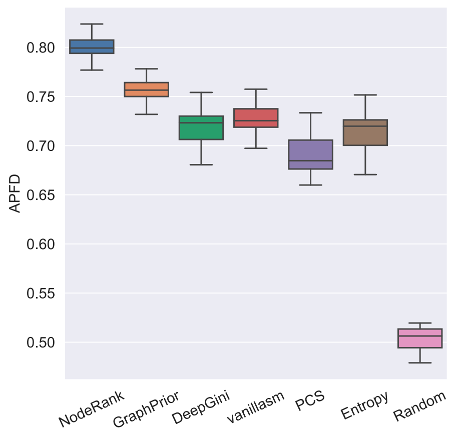


(b) PGD, LastFM, GraphSAGE

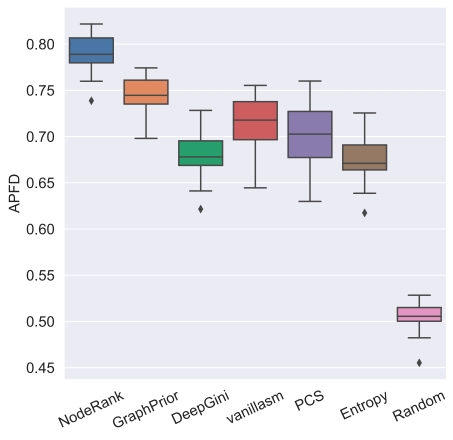
Fig. 4. Test prioritization effectiveness among NodeRank and the compared approaches for CiteSeer with GCN attacked by MMA and LastFM with GraphSAGE attacked by PGD. X-Axis: the percentage of prioritized tests; Y-Axis: the percentage of detected misclassified tests.



(a) All adversarial subjects



(b) RAR, Cora, GAT



(c) NEAA, Cora, GCN

Fig. 5. Effectiveness distributions between NodeRank and the compared approaches on adversarial test inputs.

TABLE X
CONFIDENCE INTERVAL OF NODERANK AND THE COMPARED APPROACHES IN TERMS OF APFD ON DICE-BASED GRAPH ADVERSARIAL TEST INPUTS

| Approach | Lower Bound | Upper Bound |
|---|---|---|
| Random | 0.4976 | 0.4996 |
| DeepGini | 0.6610 | 0.6693 |
| VanillaSM | 0.6848 | 0.6925 |
| PCS | 0.6715 | 0.6788 |
| Entropy | 0.6409 | 0.6491 |
| GraphPrior | 0.7271 | 0.7334 |
| **NodeRank** | 0.7686 | 0.7749 |

Note: The gray shade indicates the approach with the highest effectiveness.

in comparison to other test prioritization methods is statistically significant.

Table X presents the confidence intervals for all test prioritization methods in relation to the metric APFD. We see that NodeRank's APFD has the highest lower and upper bounds compared to other test prioritization methods. Specifically, the lower bound is 0.7686, and the upper bound is 0.7749. These experimental results underscore that in terms of APFD and considering confidence intervals, NodeRank demonstrates better effectiveness compared to other test prioritization methods.

In addition to the APFD metric, we also computed the PFD of NodeRank and compared approaches under adversarial attack scenarios, and the results are presented in Table XI and Fig. 4. As shown in Table XI, NodeRank outperformed the compared approaches regarding PFD values for all attacks and any prioritization ratio of test inputs. Notably, NodeRank detected more than 90% of the bugs when approximately 50% of the test inputs were prioritized.

Furthermore, Fig. 4 offers two visual examples for assessing the effectiveness of NodeRank compared to other approaches on the CiteSeer and LastFM datasets. In the figure, NodeRank is represented by a red line, GraphPrior by a blue line, and the baseline method by a pink line. We see that NodeRank consistently outperforms GraphPrior, as well as all confidence-based approaches and the baseline method. These experimental results demonstrate that the effectiveness of NodeRank exceeds that of all compared approaches under adversarial attack scenarios, indicating its efficacy in detecting bugs in adversarial datasets.

The box plot in Fig. 5 illustrates NodeRank's effectiveness (in terms of APFD) compared to other test prioritization methods using box plots on adversarial datasets. It presents the distribution of results from multiple repeated experiments for both NodeRank and the compared approaches. In Fig. 5, we

TABLE XI
AVERAGE COMPARISON RESULTS AMONG NODERANK AND THE COMPARED APPROACHES ON ADVERSARIAL DATA IN TERMS OF PFD

| Attack | Approach | PFD-10 | PFD-20 | PFD-30 | PFD-40 | PFD-50 | PFD-60 |
|---|---|---|---|---|---|---|---|
| DICE | Random | 0.0977 | 0.1943 | 0.2930 | 0.3959 | 0.4985 | 0.5994 |
| | DeepGini | 0.2119 | 0.3904 | 0.5359 | 0.6451 | 0.7359 | 0.8173 |
| | VanillaSM | 0.2167 | 0.4054 | 0.5601 | 0.6894 | 0.7865 | 0.8582 |
| | PCS | 0.1945 | 0.3708 | 0.5359 | 0.6704 | 0.7750 | 0.8524 |
| | Entropy | 0.2113 | 0.3835 | 0.5149 | 0.6191 | 0.7064 | 0.7843 |
| | GraphPrior | 0.2587 | 0.5023 | 0.6856 | 0.7835 | 0.8344 | 0.8923 |
| | **NodeRank** | 0.2876 | 0.5594 | 0.7569 | 0.8473 | 0.8968 | 0.9267 |
| MMA | Random | 0.1059 | 0.2126 | 0.3165 | 0.4071 | 0.5129 | 0.6036 |
| | DeepGini | 0.2261 | 0.4012 | 0.5465 | 0.6621 | 0.7572 | 0.8353 |
| | VanillaSM | 0.2379 | 0.4246 | 0.5860 | 0.6964 | 0.7885 | 0.8628 |
| | PCS | 0.2153 | 0.3976 | 0.5589 | 0.6900 | 0.7891 | 0.8649 |
| | Entropy | 0.2248 | 0.3987 | 0.5424 | 0.6567 | 0.7496 | 0.8284 |
| | GraphPrior | 0.2788 | 0.5081 | 0.6782 | 0.7823 | 0.8438 | 0.8935 |
| | **NodeRank** | 0.2972 | 0.5710 | 0.7391 | 0.8277 | 0.8801 | 0.9155 |
| NEAA | Random | 0.0974 | 0.2011 | 0.3017 | 0.3980 | 0.5045 | 0.6033 |
| | DeepGini | 0.2306 | 0.4167 | 0.5634 | 0.6724 | 0.7646 | 0.8359 |
| | VanillaSM | 0.2327 | 0.4302 | 0.5939 | 0.7215 | 0.8167 | 0.8857 |
| | PCS | 0.2022 | 0.3949 | 0.5652 | 0.6996 | 0.7977 | 0.8725 |
| | Entropy | 0.2307 | 0.4090 | 0.5342 | 0.6393 | 0.7276 | 0.7947 |
| | GraphPrior | 0.2845 | 0.5127 | 0.6943 | 0.7857 | 0.8571 | 0.9146 |
| | **NodeRank** | 0.3055 | 0.5936 | 0.7978 | 0.8811 | 0.9184 | 0.9409 |
| NEAR | Random | 0.0960 | 0.2001 | 0.2965 | 0.4005 | 0.5014 | 0.6057 |
| | DeepGini | 0.2435 | 0.4366 | 0.5813 | 0.6934 | 0.7724 | 0.8416 |
| | VanillaSM | 0.2482 | 0.4556 | 0.6148 | 0.7408 | 0.8262 | 0.8905 |
| | PCS | 0.2143 | 0.4163 | 0.5933 | 0.7219 | 0.8206 | 0.8819 |
| | Entropy | 0.2423 | 0.4274 | 0.5501 | 0.6551 | 0.7317 | 0.7954 |
| | GraphPrior | 0.3071 | 0.5788 | 0.7834 | 0.8123 | 0.8662 | 0.9014 |
| | **NodeRank** | 0.3435 | 0.6520 | 0.8266 | 0.8796 | 0.9084 | 0.9337 |
| PGD | Random | 0.0988 | 0.2033 | 0.2992 | 0.4067 | 0.5123 | 0.6161 |
| | DeepGini | 0.2350 | 0.4247 | 0.5702 | 0.6826 | 0.7706 | 0.8429 |
| | VanillaSM | 0.2487 | 0.4558 | 0.6106 | 0.7315 | 0.8162 | 0.8824 |
| | PCS | 0.2249 | 0.4209 | 0.5878 | 0.7259 | 0.8200 | 0.8845 |
| | Entropy | 0.2341 | 0.4120 | 0.5393 | 0.6432 | 0.7333 | 0.7982 |
| | GraphPrior | 0.2835 | 0.5241 | 0.6836 | 0.7826 | 0.8543 | 0.8992 |
| | **NodeRank** | 0.3171 | 0.6083 | 0.7795 | 0.8531 | 0.8996 | 0.9304 |
| RAA | Random | 0.1035 | 0.1991 | 0.2959 | 0.3985 | 0.4956 | 0.6013 |
| | DeepGini | 0.2110 | 0.3917 | 0.5392 | 0.6491 | 0.7483 | 0.8273 |
| | VanillaSM | 0.2166 | 0.4052 | 0.5686 | 0.6990 | 0.7933 | 0.8627 |
| | PCS | 0.1929 | 0.3734 | 0.5347 | 0.6755 | 0.7787 | 0.8552 |
| | Entropy | 0.2108 | 0.3862 | 0.5192 | 0.6253 | 0.7203 | 0.7960 |
| | GraphPrior | 0.2545 | 0.4523 | 0.6834 | 0.7436 | 0.8521 | 0.9034 |
| | **NodeRank** | 0.2826 | 0.5518 | 0.7587 | 0.8543 | 0.9011 | 0.9271 |
| RAF | Random | 0.1007 | 0.1997 | 0.2970 | 0.3954 | 0.4968 | 0.5925 |
| | DeepGini | 0.2108 | 0.3915 | 0.5335 | 0.6487 | 0.7412 | 0.8188 |
| | VanillaSM | 0.2158 | 0.4044 | 0.5669 | 0.6972 | 0.7921 | 0.8606 |
| | PCS | 0.1918 | 0.3728 | 0.5343 | 0.6705 | 0.7754 | 0.8516 |
| | Entropy | 0.2105 | 0.3852 | 0.5136 | 0.6228 | 0.7142 | 0.7873 |
| | GraphPrior | 0.2465 | 0.5014 | 0.6547 | 0.7362 | 0.8357 | 0.9033 |
| | **NodeRank** | 0.2816 | 0.5535 | 0.7611 | 0.8628 | 0.9048 | 0.9326 |
| RAR | Random | 0.0970 | 0.2033 | 0.3058 | 0.4053 | 0.5117 | 0.6062 |
| | DeepGini | 0.2273 | 0.4100 | 0.5573 | 0.6614 | 0.7532 | 0.8226 |
| | VanillaSM | 0.2400 | 0.4305 | 0.5854 | 0.7114 | 0.8022 | 0.8650 |
| | PCS | 0.2124 | 0.4057 | 0.5691 | 0.6981 | 0.8003 | 0.8677 |
| | Entropy | 0.2268 | 0.4044 | 0.5376 | 0.6347 | 0.7227 | 0.7909 |
| | GraphPrior | 0.2833 | 0.5344 | 0.6836 | 0.7843 | 0.8561 | 0.9013 |
| | **NodeRank** | 0.3222 | 0.6164 | 0.7853 | 0.8533 | 0.8951 | 0.9210 |

Note: The gray shade indicates the approach with the highest effectiveness.

see that, across all adversarial datasets, NodeRank's median effectiveness, as indicated by the median line within the box, surpasses that of other methods.

Regarding the quartile range, NodeRank's quartile range (i.e., the height of the box) exhibits some variations across different datasets, but overall, its upper quartile is higher than that of other methods. This difference is particularly noticeable in the subjects "RAR, Cora, GAT" and "NEAA, Cora, GCN". In terms of the outliers, we see that the box plots do not show significant outliers, indicating that NodeRank's performance across different datasets is relatively stable. Based on the above experimental results, we conclude that NodeRank outperforms all compared testing prioritization methods in terms of APFD based on the distribution of data from multiple experimental results. This demonstrates that NodeRank exhibits higher effectiveness in test prioritization compared to other methods on adversarial datasets.

TABLE XII
CONFIDENCE INTERVAL OF NODERANK AND THE COMPARED APPROACHES IN TERMS OF PFD ON ADVERSARIAL DATASETS

| **Attack** | **Approach** | **PFD-10** | | **PFD-20** | | **PFD-30** | | **PFD-40** | | **PFD-50** | | **PFD-60** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper |
| DICE | Random | 0.0951 | 0.1006 | 0.1915 | 0.1972 | 0.2903 | 0.2974 | 0.3912 | 0.3984 | 0.4948 | 0.5026 | 0.5949 | 0.6028 |
| | DeepGini | 0.2090 | 0.2160 | 0.3857 | 0.3952 | 0.5309 | 0.5392 | 0.6420 | 0.6495 | 0.7331 | 0.7397 | 0.8150 | 0.8205 |
| | VanillaSM | 0.2130 | 0.2206 | 0.4015 | 0.4090 | 0.5553 | 0.5637 | 0.6871 | 0.6922 | 0.7836 | 0.7904 | 0.8551 | 0.8626 |
| | PCS | 0.1911 | 0.1972 | 0.3675 | 0.3745 | 0.5331 | 0.5407 | 0.6667 | 0.6753 | 0.7708 | 0.7770 | 0.8475 | 0.8559 |
| | Entropy | 0.2067 | 0.2151 | 0.3801 | 0.3872 | 0.5114 | 0.5180 | 0.6161 | 0.6235 | 0.7033 | 0.7112 | 0.7814 | 0.7871 |
| | GraphPrior | 0.2542 | 0.2622 | 0.4995 | 0.5046 | 0.6835 | 0.6902 | 0.7794 | 0.7874 | 0.8303 | 0.8375 | 0.8898 | 0.8944 |
| | **NodeRank** | 0.2835 | 0.2908 | 0.5571 | 0.5618 | 0.7519 | 0.7618 | 0.8445 | 0.8511 | 0.8929 | 0.9012 | 0.9218 | 0.9306 |
| MMA | Random | 0.1019 | 0.1103 | 0.2084 | 0.2167 | 0.3115 | 0.3206 | 0.4049 | 0.4101 | 0.5094 | 0.5155 | 0.6014 | 0.6073 |
| | DeepGini | 0.2214 | 0.2285 | 0.3972 | 0.4056 | 0.5426 | 0.5510 | 0.6574 | 0.6652 | 0.7549 | 0.7599 | 0.8306 | 0.8383 |
| | VanillaSM | 0.2342 | 0.2412 | 0.4216 | 0.4279 | 0.5817 | 0.5888 | 0.6921 | 0.6997 | 0.7838 | 0.7908 | 0.8594 | 0.8676 |
| | PCS | 0.2130 | 0.2192 | 0.3940 | 0.4009 | 0.5558 | 0.5617 | 0.6856 | 0.6938 | 0.7860 | 0.7937 | 0.8602 | 0.8693 |
| | Entropy | 0.2222 | 0.2271 | 0.3951 | 0.4022 | 0.5390 | 0.5446 | 0.6537 | 0.6606 | 0.7449 | 0.7521 | 0.8246 | 0.8325 |
| | GraphPrior | 0.2755 | 0.2818 | 0.5054 | 0.5128 | 0.6760 | 0.6810 | 0.7790 | 0.7854 | 0.8388 | 0.8474 | 0.8889 | 0.8982 |
| | **NodeRank** | 0.2946 | 0.3019 | 0.5679 | 0.5752 | 0.7360 | 0.7434 | 0.8252 | 0.8304 | 0.8776 | 0.8831 | 0.9107 | 0.9193 |
| NEAA | Random | 0.0926 | 0.1010 | 0.1969 | 0.2051 | 0.2991 | 0.3038 | 0.3948 | 0.4005 | 0.4995 | 0.5073 | 0.5983 | 0.6069 |
| | DeepGini | 0.2261 | 0.2332 | 0.4130 | 0.4215 | 0.5605 | 0.5670 | 0.6703 | 0.6759 | 0.7610 | 0.7666 | 0.8323 | 0.8388 |
| | VanillaSM | 0.2286 | 0.2365 | 0.4252 | 0.4325 | 0.5891 | 0.5969 | 0.7184 | 0.7248 | 0.8142 | 0.8213 | 0.8834 | 0.8887 |
| | PCS | 0.1996 | 0.2058 | 0.3906 | 0.3991 | 0.5629 | 0.5696 | 0.6964 | 0.7029 | 0.7935 | 0.8006 | 0.8692 | 0.8761 |
| | Entropy | 0.2282 | 0.2336 | 0.4045 | 0.4134 | 0.5319 | 0.5391 | 0.6357 | 0.6415 | 0.7243 | 0.7324 | 0.7903 | 0.7996 |
| | GraphPrior | 0.2812 | 0.2866 | 0.5099 | 0.5165 | 0.6900 | 0.6968 | 0.7826 | 0.7885 | 0.8541 | 0.8615 | 0.9111 | 0.9189 |
| | **NodeRank** | 0.3018 | 0.3090 | 0.5915 | 0.5957 | 0.7930 | 0.8005 | 0.8771 | 0.8846 | 0.9140 | 0.9213 | 0.9362 | 0.9437 |
| NEAR | Random | 0.0932 | 0.0985 | 0.1956 | 0.2033 | 0.2932 | 0.3008 | 0.3980 | 0.4035 | 0.4980 | 0.5063 | 0.6022 | 0.6097 |
| | DeepGini | 0.2394 | 0.2463 | 0.4318 | 0.4408 | 0.5779 | 0.5833 | 0.6897 | 0.6963 | 0.7679 | 0.7752 | 0.8380 | 0.8437 |
| | VanillaSM | 0.2444 | 0.2530 | 0.4509 | 0.4585 | 0.6116 | 0.6179 | 0.7381 | 0.7442 | 0.8240 | 0.8300 | 0.8859 | 0.8947 |
| | PCS | 0.2105 | 0.2171 | 0.4134 | 0.4196 | 0.5908 | 0.5978 | 0.7198 | 0.7241 | 0.8180 | 0.8249 | 0.8769 | 0.8856 |
| | Entropy | 0.2389 | 0.2449 | 0.4244 | 0.4301 | 0.5468 | 0.5549 | 0.6526 | 0.6573 | 0.7288 | 0.7343 | 0.7910 | 0.7998 |
| | GraphPrior | 0.3042 | 0.3098 | 0.5740 | 0.5827 | 0.7797 | 0.7870 | 0.8093 | 0.8146 | 0.8614 | 0.8685 | 0.8979 | 0.9056 |
| | **NodeRank** | 0.3387 | 0.3467 | 0.6480 | 0.6548 | 0.8220 | 0.8295 | 0.8754 | 0.8842 | 0.9045 | 0.9114 | 0.9303 | 0.9362 |
| PGD | Random | 0.0956 | 0.1023 | 0.1984 | 0.2056 | 0.2963 | 0.3036 | 0.4032 | 0.4114 | 0.5088 | 0.5169 | 0.6124 | 0.6208 |
| | DeepGini | 0.2301 | 0.2387 | 0.4197 | 0.4276 | 0.5672 | 0.5734 | 0.6800 | 0.6867 | 0.7668 | 0.7740 | 0.8388 | 0.8472 |
| | VanillaSM | 0.2463 | 0.2531 | 0.4517 | 0.4605 | 0.6065 | 0.6148 | 0.7276 | 0.7339 | 0.8114 | 0.8187 | 0.8798 | 0.8854 |
| | PCS | 0.2222 | 0.2290 | 0.4186 | 0.4230 | 0.5851 | 0.5914 | 0.7238 | 0.7284 | 0.8161 | 0.8235 | 0.8812 | 0.8884 |
| | Entropy | 0.2306 | 0.2382 | 0.4092 | 0.4166 | 0.5347 | 0.5435 | 0.6400 | 0.6456 | 0.7287 | 0.7360 | 0.7953 | 0.8029 |
| | GraphPrior | 0.2794 | 0.2868 | 0.5193 | 0.5282 | 0.6793 | 0.6866 | 0.7793 | 0.7869 | 0.8500 | 0.8576 | 0.8967 | 0.9019 |
| | **NodeRank** | 0.3129 | 0.3205 | 0.6053 | 0.6128 | 0.7770 | 0.7842 | 0.8507 | 0.8574 | 0.8974 | 0.9023 | 0.9282 | 0.9351 |
| RAA | Random | 0.1004 | 0.1079 | 0.1949 | 0.2012 | 0.2916 | 0.2979 | 0.3960 | 0.4022 | 0.4935 | 0.4996 | 0.5981 | 0.6053 |
| | DeepGini | 0.2083 | 0.2141 | 0.3871 | 0.3944 | 0.5367 | 0.5421 | 0.6457 | 0.6533 | 0.7452 | 0.7510 | 0.8234 | 0.8318 |
| | VanillaSM | 0.2140 | 0.2188 | 0.4027 | 0.4081 | 0.5637 | 0.5727 | 0.6949 | 0.7032 | 0.7908 | 0.7959 | 0.8598 | 0.8670 |
| | PCS | 0.1894 | 0.1959 | 0.3710 | 0.3765 | 0.5310 | 0.5396 | 0.6706 | 0.6798 | 0.7748 | 0.7832 | 0.8515 | 0.8588 |
| | Entropy | 0.2080 | 0.2143 | 0.3830 | 0.3893 | 0.5161 | 0.5227 | 0.6208 | 0.6273 | 0.7164 | 0.7240 | 0.7930 | 0.8008 |
| | GraphPrior | 0.2496 | 0.2588 | 0.4491 | 0.4566 | 0.6793 | 0.6862 | 0.7403 | 0.7462 | 0.8482 | 0.8541 | 0.8991 | 0.9074 |
| | **NodeRank** | 0.2776 | 0.2875 | 0.5492 | 0.5564 | 0.7543 | 0.7626 | 0.8515 | 0.8578 | 0.8980 | 0.9036 | 0.9229 | 0.9306 |
| RAF | Random | 0.0978 | 0.1043 | 0.1971 | 0.2021 | 0.2937 | 0.2995 | 0.3913 | 0.3995 | 0.4947 | 0.5007 | 0.5898 | 0.5951 |
| | DeepGini | 0.2087 | 0.2149 | 0.3882 | 0.3946 | 0.5311 | 0.5374 | 0.6457 | 0.6517 | 0.7369 | 0.7457 | 0.8155 | 0.8224 |
| | VanillaSM | 0.2118 | 0.2197 | 0.4001 | 0.4070 | 0.5619 | 0.5691 | 0.6936 | 0.7011 | 0.7894 | 0.7944 | 0.8558 | 0.8643 |
| | PCS | 0.1875 | 0.1947 | 0.3703 | 0.3774 | 0.5307 | 0.5389 | 0.6657 | 0.6739 | 0.7732 | 0.7800 | 0.8479 | 0.8558 |
| | Entropy | 0.2062 | 0.2151 | 0.3808 | 0.3890 | 0.5104 | 0.5164 | 0.6205 | 0.6249 | 0.7101 | 0.7169 | 0.7843 | 0.7896 |
| | GraphPrior | 0.2442 | 0.2491 | 0.4973 | 0.5040 | 0.6523 | 0.6570 | 0.7319 | 0.7395 | 0.8309 | 0.8394 | 0.9008 | 0.9062 |
| | **NodeRank** | 0.2793 | 0.2842 | 0.5491 | 0.5558 | 0.7564 | 0.7654 | 0.8606 | 0.8648 | 0.9026 | 0.9076 | 0.9295 | 0.9369 |
| RAR | Random | 0.0940 | 0.1015 | 0.1992 | 0.2063 | 0.3030 | 0.3102 | 0.4031 | 0.4095 | 0.5072 | 0.5141 | 0.6023 | 0.6109 |
| | DeepGini | 0.2229 | 0.2310 | 0.4059 | 0.4137 | 0.5532 | 0.5607 | 0.6576 | 0.6642 | 0.7484 | 0.7553 | 0.8176 | 0.8249 |
| | VanillaSM | 0.2377 | 0.2449 | 0.4256 | 0.4329 | 0.5816 | 0.5902 | 0.7074 | 0.7160 | 0.7977 | 0.8044 | 0.8615 | 0.8678 |
| | PCS | 0.2099 | 0.2160 | 0.4008 | 0.4083 | 0.5656 | 0.5716 | 0.6951 | 0.7004 | 0.7953 | 0.8049 | 0.8649 | 0.8723 |
| | Entropy | 0.2236 | 0.2298 | 0.4001 | 0.4071 | 0.5350 | 0.5405 | 0.6326 | 0.6378 | 0.7182 | 0.7271 | 0.7866 | 0.7929 |
| | GraphPrior | 0.2784 | 0.2861 | 0.5320 | 0.5376 | 0.6815 | 0.6870 | 0.7809 | 0.7866 | 0.8527 | 0.8606 | 0.8983 | 0.9038 |
| | **NodeRank** | 0.3201 | 0.3254 | 0.6128 | 0.6212 | 0.7831 | 0.7875 | 0.8510 | 0.8556 | 0.8904 | 0.8977 | 0.9183 | 0.9242 |

Note: The gray shade indicates the approach with the highest effectiveness.

TABLE XIII
STATISTICAL ANALYSIS ON ADVERSARIAL DATASETS (IN TERMS OF P-VALUE UNDER THE MANN–WHITNEY U TEST)

| **Attack** | **NodeRrank vs Random** | **NodeRank vs DeepGini** | **NodeRank vs VanillaSM** | **NodeRank vs PCS** | **NodeRank vs Entropy** | **NodeRank vs GraphPrior** |
|---|---|---|---|---|---|---|
| DICE | $3.3978 \times 10^{-8}$ | $6.8803 \times 10^{-7}$ | $1.2979 \times 10^{-5}$ | $2.9367 \times 10^{-6}$ | $1.7078 \times 10^{-7}$ | 0.0021 |
| MMA | $2.4541 \times 10^{-8}$ | $6.4702 \times 10^{-5}$ | $4.6045 \times 10^{-4}$ | $1.2353 \times 10^{-4}$ | $2.0828 \times 10^{-5}$ | 0.0014 |
| NEAA | $4.5746 \times 10^{-8}$ | $3.3978 \times 10^{-7}$ | $2.3978 \times 10^{-5}$ | $4.2542 \times 10^{-6}$ | $1.4537 \times 10^{-7}$ | 0.0002 |
| NEAR | $2.6732 \times 10^{-8}$ | $6.1731 \times 10^{-7}$ | $1.1088 \times 10^{-6}$ | $5.3228 \times 10^{-7}$ | $3.3978 \times 10^{-7}$ | 0.0009 |
| PGD | $3.3274 \times 10^{-8}$ | $2.6274 \times 10^{-5}$ | $4.6045 \times 10^{-4}$ | $5.1867 \times 10^{-5}$ | $1.3448 \times 10^{-6}$ | 0.0017 |
| RAA | $8.2331 \times 10^{-8}$ | $3.4582 \times 10^{-7}$ | $5.5223 \times 10^{-6}$ | $1.7497 \times 10^{-6}$ | $1.1088 \times 10^{-7}$ | 0.0016 |
| RAF | $3.4582 \times 10^{-8}$ | $1.0307 \times 10^{-6}$ | $1.4624 \times 10^{-5}$ | $2.2701 \times 10^{-6}$ | $1.7078 \times 10^{-7}$ | 0.0057 |
| RAR | $6.4691 \times 10^{-8}$ | $3.9739 \times 10^{-7}$ | $5.5223 \times 10^{-6}$ | $1.3448 \times 10^{-6}$ | $9.5885 \times 10^{-7}$ | 0.0041 |

Moreover, Tables XII and XIV displays the confidence intervals of all test prioritization methods. Table XII displays the confidence intervals of all test prioritization methods in terms of PFD. We see that, in terms of PFD, NodeRank also demonstrates the highest lower and upper bounds compared to other test prioritization approaches when prioritizing different ratios

TABLE XIV
CONFIDENCE INTERVAL OF NODERANK AND THE COMPARED APPROACHES IN TERMS OF APFD ON ADVERSARIAL TEST INPUTS

| Approach | Lower Bound | Upper Bound |
|---|---|---|
| Random | 0.5013 | 0.5022 |
| DeepGini | 0.6705 | 0.6738 |
| VanillaSM | 0.6958 | 0.6987 |
| PCS | 0.6832 | 0.6861 |
| Entropy | 0.6501 | 0.6534 |
| GraphPrior | 0.7319 | 0.7344 |
| **NodeRank** | 0.7760 | 0.7783 |

Note: The gray shade indicates the approach with the highest effectiveness.

of tests. These experimental findings emphasize that, from the perspective of confidence intervals, NodeRank shows higher effectiveness compared to other test prioritization methods.

Table XIV displays the confidence intervals in terms of APFD. In Table XIV, NodeRank's APFD shows the highest lower and upper bounds compared to other test prioritization methods, with values of 0.7760 and 0.7783, respectively. Remarkably, NodeRank's lower bound (0.7760) even surpasses the upper bounds of all other comparative methods. GraphPrior's upper bound is 0.7344, while the upper bounds for other test prioritization methods range from 0.5022 to 0.6987.

**Answer to RQ2:** On adversarial test inputs, NodeRank consistently demonstrates better effectiveness in comparison to GraphPrior, all confidence-based approaches, and the baseline method across all subjects in terms of both the APFD and PFD metrics. Regarding APFD, NodeRank exhibits an average improvement of 4.96% and 62.15% over the compared methods.

### C. RQ3: Influence of Ensemble Learning Methods

**Objective.** We investigate the impact of ensemble learning strategies on NodeRank's effectiveness in the learning-to-rank process.

**Experimental design.** We employ NodeRank and its variants, namely NodeRank$^V$ and NodeRank$^S$ (cf. Section IV-F for details), to prioritize test inputs for both natural and adversarial scenarios, and evaluate their effectiveness in terms of APFD. These variants differ in the ensemble learning strategies used in the learning-to-rank process.

**Results.** Table XV presents the average effectiveness of NodeRank and its variants, along with several compared approaches, on both natural and adversarial datasets. The upper part shows the average effectiveness under different models, while the bottom part shows the average effectiveness across different datasets. From Table XV, we can observe that the average effectiveness of NodeRank and its variants outperform all the compared approaches (i.e., GraphPrior, the confidence-based approaches and random selection) in each case. Additionally, the effectiveness of NodeRank is comparatively better than their variants. Across different GNN models, NodeRank performs the best in 100% of the cases on natural data. Furthermore, on the adversarial data, NodeRank also outperforms in 100% of the cases. From the perspective of datasets, on natural data, NodeRank performs better than all the variants in each case. On adversarial data, NodeRank has the highest average effectiveness across all adversarial datasets. Overall, the final average effectiveness of NodeRank is 0.7833 and 0.7772 on natural and adversarial datasets, respectively. These experimental results demonstrate that the sum-based ensemble learning strategies used in NodeRank is more suitable for test prioritization.

**Answer to RQ3:** On both natural and adversarial datasets, NodeRank offers a better effectiveness, in terms of APFD, over other variants. We also note that any variant of NodeRank outperforms all the compared approaches in GNN test prioritization.

### D. RQ4: Ablation Study of Mutation Operators

**Objective:** We investigate the effect of each category of mutation operators (i.e., GSM, NFM, and GMM). To this end, we analyze the contributions of the features generated by each type of mutation operator and conduct corresponding ablation studies. We proceed as proposed by Meyes et al. [53]: We measure the impact of a component on an ML system by removing or replacing this component and observing whether the performance of the ML system is affected. The objective is not to comprehensively check which feature set combinations provide good performance but rather to check that each set contributes to the performance.

**Experimental design:** We assume that the node mutation features (NFM) as a key component of the NodeRank approach. Then, the graph structure mutation (GSM) features, which are obtained from the dataset, are considered the next most important feature set. Finally, the graph model mutation (GMM) features are considered as the first that can be removed in the ablation study, following the process in [53]. The experimental steps for checking the contributions of each subset of mutation features to the performance of NodeRank are thus as follows:

1) We compute the test prioritization performance of NodeRank when all mutation features are used.
2) We compute the test prioritization performance of a variant of NodeRank where the ranking model is learned with vectors that do not consider GMM features.
3) We compute the test prioritization performance of a variant of NodeRank where the ranking model is learned only with NFM feature vectors (i.e., by removing the GMM and GSM).
4) Finally, we also consider the case where no features are used. NodeRank, therefore, does not implement ensemble learning to rank. Instead, we consider a random ranking approach to prioritize the test set.

Note that we do not attempt to perform experiments that compare the value of the different feature sets. Indeed, the mutation space of GNNs is complex, and mutations of different types can produce feature vectors of various sizes, which may implicitly impact the learning performance, making any performance comparison biased or uninformative.

**Results:** The results of the ablation experiment are reported in Table XVI. As expected, the Random prioritization approach,

TABLE XV
PERFORMANCE (APFD SCORES) OF NODERANK VARIANTS ASSOCIATED TO DIFFERENT ENSEMBLE LEARNING STRATEGIES (#BC ⇔ #BEST CASES) AND (AVG ⇔ AVERAGE APFD SCORE)

| Approach | Natural inputs | | | | | Adversarial inputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | #BC | GAT | GCN | GraphSAGE | TAGCN | #BC | GAT | GCN | GraphSAGE | TAGCN |
| Random | 0 | 0.4864 | 0.5028 | 0.4967 | 0.4959 | 0 | 0.5037 | 0.5023 | 0.5009 | 0.5014 |
| DeepGini | 0 | 0.6353 | 0.6805 | 0.7144 | 0.6855 | 0 | 0.6325 | 0.6737 | 0.7115 | 0.6850 |
| VanillaSM | 0 | 0.6792 | 0.7134 | 0.7236 | 0.7120 | 0 | 0.6686 | 0.7045 | 0.7202 | 0.7089 |
| PCS | 0 | 0.6789 | 0.7069 | 0.7048 | 0.7020 | 0 | 0.6610 | 0.6934 | 0.7003 | 0.6961 |
| Entropy | 0 | 0.6174 | 0.6650 | 0.6950 | 0.6474 | 0 | 0.6167 | 0.6607 | 0.6921 | 0.6477 |
| GraphPrior | 0 | 0.7475 | 0.7421 | 0.7501 | 0.7463 | 0 | 0.7298 | 0.7307 | 0.7428 | 0.7354 |
| NodeRank$^V$ | 0 | 0.7607 | 0.7505 | 0.7505 | 0.7481 | 0 | 0.7537 | 0.7483 | 0.7558 | 0.7475 |
| NodeRank$^S$ | 0 | 0.7551 | 0.7495 | 0.7512 | 0.7561 | 0 | 0.7516 | 0.7447 | 0.7527 | 0.7533 |
| **NodeRank** | 16 | 0.7876 | 0.7761 | 0.7851 | 0.7846 | 108 | 0.7795 | 0.7731 | 0.7872 | 0.7802 |

| Approach | with Natural inputs | | | | | with Adversarial inputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CiteSeer | Cora | LastFM | Pubmed | Avg | CiteSeer | Cora | LastFM | Pubmed | Avg |
| Random | 0.4849 | 0.4954 | 0.5011 | 0.5004 | 0.4955 | 0.5028 | 0.5064 | 0.4990 | 0.4991 | 0.5018 |
| DeepGini | 0.6315 | 0.7209 | 0.7017 | 0.6615 | 0.6788 | 0.6277 | 0.7082 | 0.6927 | 0.6604 | 0.6722 |
| VanillaSM | 0.6623 | 0.7397 | 0.7415 | 0.6847 | 0.7071 | 0.6524 | 0.7244 | 0.7318 | 0.6807 | 0.6973 |
| PCS | 0.6671 | 0.7251 | 0.7348 | 0.6656 | 0.6981 | 0.6511 | 0.7043 | 0.7257 | 0.6578 | 0.6847 |
| Entropy | 0.6279 | 0.7154 | 0.6183 | 0.6632 | 0.6562 | 0.6253 | 0.7027 | 0.6173 | 0.6618 | 0.6518 |
| GraphPrior | 0.6842 | 0.7801 | 0.7821 | 0.7448 | 0.7478 | 0.6735 | 0.7624 | 0.7637 | 0.7332 | 0.7332 |
| NodeRank$^V$ | 0.6845 | 0.7873 | 0.7909 | 0.7471 | 0.7525 | 0.6941 | 0.7745 | 0.7839 | 0.7397 | 0.7481 |
| NodeRank$^S$ | 0.6917 | 0.7845 | 0.7893 | 0.7465 | 0.7532 | 0.6881 | 0.7742 | 0.7864 | 0.7398 | 0.7471 |
| **NodeRank** | 0.7261 | 0.8158 | 0.8146 | 0.7768 | 0.7833 | 0.7286 | 0.8011 | 0.8093 | 0.7690 | 0.7772 |

Note: The gray shade indicates the approach with the highest effectiveness.

TABLE XVI
FEATURE ABLATION STUDY RESULTS

| Approach | with Natural inputs | | | | | with Adversarial inputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CiteSeer | Cora | LastFM | Pubmed | Avg | CiteSeer | Cora | LastFM | Pubmed | Avg |
| Random prioritization (b/c No features) | 0.4849 | 0.4954 | 0.5011 | 0.5004 | **0.4955** | 0.5028 | 0.5064 | 0.4990 | 0.4991 | **0.5018** |
| DeepGini | 0.6315 | 0.7209 | 0.7017 | 0.6615 | **0.6788** | 0.6277 | 0.7082 | 0.6927 | 0.6604 | **0.6722** |
| VanillaSM | 0.6623 | 0.7397 | 0.7415 | 0.6847 | **0.7071** | 0.6524 | 0.7244 | 0.7318 | 0.6807 | **0.6973** |
| PCS | 0.6671 | 0.7251 | 0.7348 | 0.6656 | **0.6981** | 0.6511 | 0.7043 | 0.7257 | 0.6578 | **0.6847** |
| Entropy | 0.6279 | 0.7154 | 0.6183 | 0.6632 | **0.6562** | 0.6253 | 0.7027 | 0.6173 | 0.6618 | **0.6518** |
| GraphPrior | 0.6842 | 0.7801 | 0.7821 | 0.7448 | **0.7478** | 0.6735 | 0.7624 | 0.7637 | 0.7332 | **0.7332** |
| NodeRank$_{NFM}$ | 0.5828 | 0.6328 | 0.5933 | 0.6085 | **0.6044** | 0.5710 | 0.6200 | 0.5897 | 0.6030 | **0.5959** |
| NodeRank$_{NFM+GSM}$ | 0.6315 | 0.6796 | 0.7025 | 0.6452 | **0.6647** | 0.6569 | 0.6876 | 0.7106 | 0.6504 | **0.6764** |
| **NodeRank** (i.e., NodeRank$_{NFM+GSM+GMM}$) | 0.7261 | 0.8158 | 0.8146 | 0.7768 | **0.7833** | 0.7286 | 0.8011 | 0.8093 | 0.7690 | **0.7772** |

Note: The gray shade indicates the approach with the highest effectiveness.

which employs no mutation features for learning to rank, performs the worst in terms of APFD. In contrast, the NodeRank approach that learns to rank by incorporating all three mutation rule sets (pertaining to nodes, graph structure, and graph model) exhibits the highest performance. Remarkably, the exclusion of graph model mutation features leads to a decline in learning performance by approximately 17.84% and 14.90% in terms of APFD on natural and adversarial datasets, respectively. On the other hand, employing only node mutation features yields a significant improvement over Random prioritization, with a performance gain of approximately 21.98% and 18.75% in terms of APFD on natural and adversarial datasets, respectively.

Moreover, by comparing against the performance of uncertainty-based DNN test prioritization approaches and GraphPrior, we note that the combinations of the three categories of mutation features were necessary to achieve state-of-the-art performance in GNN test prioritization.

**Answer to RQ4:** The design choice in NodeRank to include all three types of mutation operators was effective. Indeed, although the node mutation operator can enable NodeRank to outperform random prioritization, it is the combination of NFM, GSM, and GMM operators that together lead to the SOTA performance of NodeRank.

## *E. RQ5: Investigating the Contributions of Model Mutation Rules on NodeRank Effectiveness*

**Objective:** In this research question, our aim is to demonstrate that the model mutation rules of NodeRank actually contribute to its effectiveness. In the original NodeRank, we utilize the killing approaches in traditional mutation analysis for DNNs. This killing process is used to generate model mutation features of a given test input. The features are then utilized to predict the misclassification probability of this input. In this process, the model mutation features generated by killing may contain information resulting from model mutation rules and randomness in model training, both of which may contribute to the effectiveness of NodeRank. In this research question, by utilizing the killing approach in DeepCrime [21], which considers the training randomness in the process of the killing, we aim to demonstrate that the model mutation rules actually contribute to the effectiveness of NodeRank.

**Experimental design:** To demonstrate the aforementioned objective, we designed three types of variants of NodeRank: 1) $\text{NodeRank}_{\text{DeepCrime}}$, a variant that utilizes DeepCrime's killing method to mitigate the influence of randomness when generating model mutation features. DeepCrime's killing approach takes into account the training randomness of the mutated model. Specifically, this killing approach requires repeating the training process $n$ times for both the original model $N = \langle N_1, \ldots, N_n \rangle$ and its mutated model $M = \langle M_1, \ldots, M_n \rangle$. A test is considered killed if the difference between the outputs of the original and mutated models is statistically significant with non-negligible and non-small effect size. 2) $\text{NodeRank}_{\text{Random}}$, which does not incorporate model mutation rules and solely relies on random generation of model mutation features, and 3) $\text{NodeRank}_{\text{withoutGMM}}$, which does not utilize model mutation features. We validated whether model mutation rules contribute to the effectiveness of NodeRank by comparing the effectiveness of these three variants. If $\text{NodeRank}_{\text{DeepCrime}}$ outperformed both $\text{NodeRank}_{\text{Random}}$ and $\text{NodeRank}_{\text{withoutGMM}}$, we consider that the model mutation rules contribute to the effectiveness of NodeRank.

In the subsequent sections, we first describe the detailed implementation of DeepCrime. Then, we present the details of the variants of NodeRank and how we leverage these variants to demonstrate that model mutation rules contribute to NodeRank's effectiveness.

### 1) Implementation of DeepCrime

Given an original GNN model $N$ and a test $t$, the DeepCrime approach follows the following method to determine whether a test is "killed".

❶ For the original GNN model $N$, we repeated its training process $n$ times, resulting in $n$ GNN models: $\langle N_1, \ldots, N_n \rangle$. Similarly, for the mutated model $M$, we repeated its training process $n$ times, obtaining $\langle M_1, \ldots, M_n \rangle$. Consistent with previous research [21], we set $n = 20$ in our experiments.

❷ For $\langle N_1, \ldots, N_n \rangle$, we used each GNN model to make predictions on the test $t$ and obtained the predicted classifications for $t$ from each model. Similarly, for $\langle M_1, \ldots, M_n \rangle$, we used each mutated model to predict the test $t$ and obtained the predicted classifications for $t$ from each model.

❸ Prior work [21] suggests that the mutated model $M$ is considered "killed" if, for the given test $t$, the difference between the output of the original and mutated models, denoted as $A_N(t) = \langle A_{N_1}, \ldots, A_{N_n} \rangle$ and $A_M(t) = \langle A_{M_1}, \ldots, A_{M_n} \rangle$, is statistically significant with a non-negligible and non-small effect size. Therefore, we measure whether the mutated model $M$ is "killed" using Formula 12.

$$isKill(N, M, t) = \begin{cases} \text{true} & \text{if effectSize}(A_N(\text{t}), A_M(\text{t})) \geq \beta \\ & \text{and } p\_\text{value}(A_N(\text{t}), A_M(\text{t})) < \alpha \\ \text{false} & \text{otherwise} \end{cases} \tag{12}$$

In Formula 12, $isKilled$ indicates whether the test $t$ "kills" the mutated model $M$. $A_N(t)$ represents a series of predictions (outputs) for test $t$ from the set of models $\langle N_1, \ldots, N_n \rangle$. Similarly, $A_M(t)$ refers to $\langle A_{M_1}, \ldots, A_{M_n} \rangle$, representing the set of predictions (outputs) for test $t$ from the set of models $\langle M_1, \ldots, M_n \rangle$.

The term "effect size" [73] quantitatively measures the difference between two distributions of APFD results. One commonly used measure of effect size is Cohen's $d$. This value can be interpreted using thresholds provided by Cohen [74]: $|d| < 0.2$ indicates a "negligible" effect, $|d| < 0.5$ indicates a "small" effect, $|d| < 0.8$ indicates a "medium" effect, and otherwise, it is considered a "large" effect. The prior study on DNN mutation analysis [21] pointed out that the effect size should be non-small. The $\beta$ can have an impact on the effectiveness of the killing method DeepCrime in the context of NodeRank for test prioritization. In our experiments, we set $\beta$ to be 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9, covering a wide range of effect sizes. By adopting different effect size values, we can observe whether, under different effect sizes, the effectiveness of the NodeRank variants using DeepCrime and mutation rules is better than the variant not using mutation rules but instead randomly generating mutated models, thereby better validating the contributions of model mutation rules.

### 2) Variants of NodeRank

In the following, we explain how we design variants of NodeRank and how we utilize them to demonstrate the effectiveness of NodeRank's model mutation rule.

❶ In the first step, we generate three types of variants of NodeRank: $\text{NodeRank}_{\text{withoutGMM}}$, $\text{NodeRank}_{\text{Random}}$, and $\text{NodeRank}_{\text{DeepCrime}}$. Regarding $\text{NodeRank}_{\text{DeepCrime}}$, we set different values for the effect size, ranging from 0.3 to 0.9. This aims to measure the effectiveness of $\text{NodeRank}_{\text{DeepCrime}}$ across varying effect sizes, providing a clearer demonstration of the effectiveness of our model mutation rules.

❷ $\text{NodeRank}_{\text{withoutGMM}}$ does not utilize model mutation features for test prioritization. All other workflow processes in this variant remain consistent with the original NodeRank.

❸ $\text{NodeRank}_{\text{Random}}$ utilizes model mutation features for test prioritization. However, the generated mutated models do not correspond to actual mutations; instead, it chooses to obtain different but equivalent GNN models as mutated models. Due to the randomness in training GNN models (such as the random initialization of model weights before training), different initializations can lead to different optimization paths during training, resulting in different weights at the end of training. Therefore, under the same configuration and operating conditions, the generated models can vary. In $\text{NodeRank}_{\text{Random}}$, we generated a series of

TABLE XVII
EFFECTIVENESS (APFD SCORES) OF NODERANK'S VARIANTS (NODERANK$_{withoutGMM}$ DOES NOT GENERATE MUTATED MODELS. NODERANK$_{Random}$ DOES NOT USE MODEL MUTATION RULES TO GENERATE MUTATED MODELS. NODERANK$_{DeepCime}$ USES MODEL MUTATION RULES TO GENERATE MUTATED MODELS)

| Approach | Natural inputs | | | | | Adversarial inputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | #BC | GAT | GCN | GraphSAGE | TAGCN | #BC | GAT | GCN | GraphSAGE | TAGCN |
| NodeRank$_{withoutGMM}$ | 0 | 0.6607 | 0.6467 | 0.6701 | 0.6812 | 0 | 0.6772 | 0.6634 | 0.6826 | 0.6965 |
| NodeRank$_{Random}$ | 0 | 0.6892 | 0.7143 | 0.7185 | 0.7162 | 0 | 0.6847 | 0.7112 | 0.7152 | 0.7144 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.3) | 0 | 0.7465 | 0.7448 | 0.7571 | 0.7597 | 0 | 0.7440 | 0.7416 | 0.7575 | 0.7542 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.4) | 0 | 0.7472 | 0.7445 | 0.7573 | 0.7599 | 14 | 0.7439 | 0.7417 | 0.7569 | 0.7543 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.5) | 8 | 0.7480 | 0.7449 | 0.7558 | 0.7602 | 7 | 0.7439 | 0.7418 | 0.7571 | 0.7542 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.6) | 0 | 0.7472 | 0.7444 | 0.7568 | 0.7601 | 38 | 0.7441 | 0.7420 | 0.7572 | 0.7541 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.7) | 2 | 0.7444 | 0.7402 | 0.7605 | 0.7603 | 15 | 0.7425 | 0.7398 | 0.7576 | 0.7540 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.8) | 2 | 0.7414 | 0.7353 | 0.7571 | 0.7622 | 21 | 0.7412 | 0.7372 | 0.7557 | 0.7540 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.9) | 4 | 0.7383 | 0.7331 | 0.7573 | 0.7601 | 13 | 0.7398 | 0.7338 | 0.7533 | 0.7537 |

| Approach | with Natural inputs | | | | | with Adversarial inputs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CiteSeer | Cora | LastFM | Pubmed | Avg | CiteSeer | Cora | LastFM | Pubmed | Avg |
| NodeRank$_{withoutGMM}$ | 0.6315 | 0.6796 | 0.7025 | 0.6452 | 0.6647 | 0.6569 | 0.6876 | 0.7106 | 0.6504 | 0.6764 |
| NodeRank$_{Random}$ | 0.6686 | 0.7382 | 0.7452 | 0.6859 | 0.7095 | 0.6619 | 0.7460 | 0.7498 | 0.6676 | 0.7063 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.3) | 0.6995 | 0.7785 | 0.7734 | 0.7565 | 0.7520 | 0.7012 | 0.7677 | 0.7694 | 0.7495 | 0.7470 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.4) | 0.6998 | 0.7787 | 0.7736 | 0.7568 | 0.7521 | 0.7017 | 0.7667 | 0.7694 | 0.7495 | 0.7469 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.5) | 0.6989 | 0.7794 | 0.7739 | 0.7567 | 0.7522 | 0.7012 | 0.7670 | 0.7695 | 0.7496 | 0.7468 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.6) | 0.6993 | 0.7787 | 0.7737 | 0.7568 | 0.7521 | 0.7015 | 0.7672 | 0.7696 | 0.7497 | 0.7471 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.7) | 0.6991 | 0.7797 | 0.7706 | 0.7560 | 0.7513 | 0.7016 | 0.7669 | 0.7682 | 0.7478 | 0.7461 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.8) | 0.6967 | 0.7753 | 0.7692 | 0.7547 | 0.7490 | 0.7005 | 0.7664 | 0.7659 | 0.7457 | 0.7446 |
| NodeRank$_{DeepCrime}$(effectSize $\geq$ 0.9) | 0.6965 | 0.7746 | 0.7661 | 0.7515 | 0.7472 | 0.6991 | 0.7646 | 0.7640 | 0.7432 | 0.7427 |

Note: The gray shade indicates the approach with the highest effectiveness.

equivalent GNN models as mutated models using the same configuration and operating conditions. All other workflow processes in this variant remain consistent with the original NodeRank.

❹ NodeRank$_{\text{DeepCrime}}$ uses DeepCrime's mutation killing approach as the killing approach to generate model mutation features for test prioritization. All other workflow processes in this variant remain consistent with the original NodeRank.

In summary, NodeRank$_{\text{withoutGMM}}$ represents NodeRank without the use of model mutation features. NodeRank$_{\text{Random}}$ incorporates model mutation features, but the mutated models are not generated by model mutation rules; instead, they alter the initial random seed to produce different but equivalent GNN models as the mutated models. NodeRank$_{\text{DeepCrime}}$ utilizes model mutation rules to generate mutated models. After completing the above process, we consider that if the effectiveness of NodeRank$_{\text{DeepCrime}}$ is higher than that of NodeRank$_{\text{Random}}$ and NodeRank$_{\text{withoutGMM}}$, the model mutation rules operated in NodeRank$_{\text{DeepCrime}}$ contribute to its effectiveness.

**Results:** Table XVII presents the experimental results for RQ5. We highlighted the approach with the highest effectiveness in grey to facilitate quick and easy interpretation of the results. The table above showcases the average effectiveness of NodeRank$_{\text{withoutGMM}}$, NodeRank$_{\text{Random}}$, and NodeRank$_{\text{DeepCrime}}$ across different subjects in terms of models. The table below shows the average effectiveness in terms of datasets.

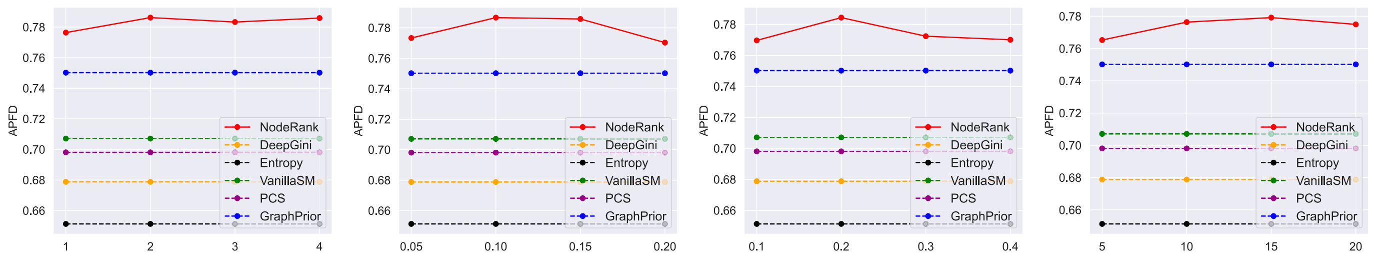
The two tables show that in each case, NodeRank$_{\text{DeepCrime}}$ performs the best. Furthermore, each variant of NodeRank$_{\text{DeepCrime}}$, regardless of different effect size settings, exhibits higher effectiveness than that of NodeRank$_{\text{Random}}$. According to the experimental design mentioned above, if the effectiveness of NodeRank$_{\text{DeepCrime}}$ is higher than that of NodeRank$_{\text{Random}}$ and NodeRank$_{\text{withoutGMM}}$, we consider that the model mutations (generated by model mutation rules) in NodeRank$_{\text{DeepCrime}}$ contribute to mutated models. Therefore, the above experimental results indicate that the model mutation rule of NodeRank actually contributes to its effectiveness.

**Answer to RQ5:** The model mutation rules of NodeRank actually contribute to its effectiveness.

## *F. RQ6: Influence of Mutation Operator Parameters on NodeRank*

**Objective:** In NodeRank, we designed a set of new mutation operators specifically for GNNs. In this research question, we explore the influence of the parameter ranges of mutation operators on NodeRank.

**Experimental design:** First, we selected multiple mutation operators with parameters of integer/float types. This choice was made because, for Boolean-type mutation operators, mutations involve toggling between True and False, resulting in only one possible parameter value, rendering parameter changes unfeasible. Following the approach from the existing study [11], for each investigated mutation operator, we systematically varied its parameters multiple times while keeping the parameters of other mutation operators in their initial states. Subsequently, we recorded NodeRank's effectiveness (measured by APFD) after each parameter change. We used line graphs to visually depict the impact of parameter changes on NodeRank's effectiveness for each mutation operator.



(a) Number of edges (b) Node feature offset (c) Negative slope (d) Hidden channel

Fig. 6. Impact of mutation operator parameters in NodeRank.

Specifically, NodeRank consists of three types of mutation operators: Graph structure mutation (GSM), Node feature mutation (NFM), and GNN model mutation (GMM). Since these mutation operators aim to introduce subtle modifications to the original test set or the GNN model, we aim to ensure that after adjusting parameter ranges, the new parameter values also result in relatively slight changes.

- **Graph structure mutation (GSM)** GSM includes a mutation operator that involves slightly changing the structure of the input graph by randomly adding edges. Consequently, the parameter for this mutation operator is *the number of added edges*. This parameter was set to 1, 2, 3, and 4 to investigate the impact of the parameter range of this mutation operator on the effectiveness of NodeRank.
- **Node feature mutation (NFM)** NFM includes a mutation operator that changes the features of the targeted nodes to adjust their positions in the feature space. Consequently, the parameter for NFM is *the node feature offset*. This parameter was set to 0.05, 0.10, 0.15, and 0.20 to investigate the impact of the parameter range of this mutation operator on the effectiveness of NodeRank.
- **GNN model mutation (GMM)** GMM comprises multiple mutation operators that aim to make slight changes to the training parameters in NodeRank. We selected mutation operators with integer/float-type parameters and adjusted their parameter ranges. These include "*Negative Slope*" with parameter range adjustments of (0.1, 0.2, 0.3, 0.4) and "*Hidden Channel*" with parameter ranges of (5, 10, 15, 20).

**Results:** The experimental results of RQ6 are presented in Fig. 6. The experiments are conducted on 16 natural subjects. Among them, Fig. 6(a) shows the impact of changing the parameter number of edges for the mutation operator targeting graph structure. Fig. 6(b) illustrates the influence of the parameter node feature offset for the mutation operator targeting node features. Fig. 6(c) demonstrates the impact of the parameter negative slope for the mutation operator targeting the GNN models. Fig. 6(d) displays the influence of the parameter Hidden channel for the mutation operator targeting the GNN model. In this context, the red line represents NodeRank. First, we see that across all parameter settings of the mutation operator, NodeRank effectiveness consistently exceeds that of all the comparative test prioritization methods (i.e., GraphPrior, confidence-based approaches and random selection). Moreover, we found that NodeRank performs stably when the parameter values of the newly designed mutation operators change. For example, when modifying the "Number of edges" parameter, the APFD values of NodeRank vary within the range of approximately 0.778 to 0.785. Similarly, when adjusting the "Node feature offset" parameter, the APFD values of NodeRank fluctuate between approximately 0.777 and 0.785.

**Answer to RQ6:** Across all parameter settings of the newly designed mutation operator, NodeRank's effectiveness consistently outperforms that of other comparative test prioritization methods. Moreover, the effectiveness of NodeRank remains stable when the parameter values change.

## VI. Discussion

### A. Generality of NodeRank

Our proposed NodeRank and its variants perform test prioritization for GNNs via ensemble-learning-based mutation analysis. The evaluation on 124 subjects demonstrates their effectiveness on both natural and adversarial datasets. The scheme of NodeRank, (i.e., slightly changing graph inputs and graph models) can also be generalized to edge-level and graph-level GNN tasks. In the future, we will carefully design relevant mutation rules to further adapt NodeRank to other GNN tasks.

Additionally, we discuss the potential applicability of NodeRank for regression tasks. However, currently, the mutation rules and ranking models of NodeRank are designed explicitly for classification tasks. To extend NodeRank to regression tasks, modifications to the model mutation rules and ranking models would be required. If appropriate model mutation rules can be identified for regression tasks and suitable ranking models can be designed, NodeRank could also be a promising approach for regression tasks.

### B. Challenges of NodeRank

NodeRank requires a sufficiently large training set to train its internal ranking model. This training set includes labels (i.e., samples that the model predicts incorrectly are labeled as 1, while correctly predicted samples are labeled as 0). If the original model has very high accuracy, it can result in very few training samples labeled as 1, potentially leading to an imbalanced dataset during the training of NodeRank's ranking model.

An imbalanced dataset can cause a decrease in performance when dealing with samples labeled as 1, as there are not enough examples to learn how to rank these samples correctly.

For example, in a scenario involving bank transfer transactions, where each account represents a node and edges represent transfer transactions between accounts, GNN models can be used to identify fraudulent accounts (i.e., whether a node is a fraudulent account or not). If the GNN model has a very high accuracy (few nodes predicted incorrectly), it will result in very few samples labeled as 1 in the NodeRank training set. This directly affects the training of the ranking model in NodeRank. Under these conditions, the effectiveness of NodeRank in prioritizing the misclassified accounts will be affected.

### C. Differences in Approaches for NodeRank

In this Section, we discuss the differences in approaches for NodeRank from three perspectives, namely the differences in evaluating NodeRank methods, the differences between NodeRank and its variants, as well as different NodeRank approaches with different types of features.

[*Differences in evaluating NodeRank methods*] In addition to evaluating NodeRank on natural datasets, we assess its effectiveness from three different perspectives, as presented in RQ2 through RQ4. This is because these perspectives cover key aspects and contribute to a comprehensive understanding of NodeRank's performance. In RQ2, we assess the efficacy of NodeRank when confronted with adversarial test inputs. In RQ3, we explore how ensemble learning strategies influence NodeRank's effectiveness within the context of learning-to-rank. In RQ4, we examine the individual contributions of each category of mutation features (GSM, NFM, and GMM) that are generated for NodeRank's learning-to-rank model. Below, we provide a detailed explanation of the differences across approaches for RQ2, RQ3, and RQ4, as well as why it is important to assess the effectiveness of NodeRank from these three different perspectives.

- **RQ2 - Evaluation on Adversarial Test Inputs** This perspective focuses on assessing NodeRank's performance when confronted with adversarial test inputs. It is critical because it reveals NodeRank's resilience and reliability in handling challenging input data. In contrast to the evaluation methods in RQ2 and RQ3, this assessment is conducted using adversarial test inputs rather than natural datasets.
- **RQ3 - Impact of Ensemble Learning Strategies** This perspective investigates how different ensemble learning strategies influence NodeRank's effectiveness within the context of learning-to-rank. This investigation is significant as it helps us understand which strategies are more suitable for NodeRank to perform test prioritization.
- **RQ4 - Contributions of Mutation Features:** In RQ4, we delve into the individual contributions of each category of mutation features (GSM, NFM, and GMM) on NodeRank. Understanding these differences is essential to identify which features are most critical for NodeRank's effectiveness, guiding further research and development efforts.

[*Differences between NodeRank and its variants*] In RQ3, we propose several variants of NodeRank. In RQ3, the variants of NodeRank differ in the ensemble learning strategies used to combine base ranking models. Apart from this distinction, the workflows of the NodeRank variants remain identical to NodeRank.

[*Different NodeRank approaches with different types of features*] In RQ4, we design different NodeRank approaches, which apply different types of mutation features for test prioritization. Specifically, NodeRank$_{NFM}$ only applies the NFM features. NodeRank$_{NFM+GSM}$ applies both the NFM and GSM features. Our aim is to investigate the contributions of each feature type to the effectiveness of NodeRank.

### D. Threats to Validity

*Threats to Internal Validity.* The internal threat to validity mainly exists in the implementation of NodeRank, its variants, and the compared approaches. To reduce the threat, we implemented all approaches based on the widely used library PyTorch. Concerning the compared test prioritization approaches, we considered the implementations released by the authors. Another internal threat lies in the randomness of the model training process. To mitigate this threat, we conducted a statistical analysis involving performing ten repetitions of the model training process for both the original and mutated models. We then used these results to calculate the statistical significance of the experiments. The selection of the mutation rules used in our study represents another potential threat to the internal validity of our research. Despite our best efforts to identify model mutation rules, it is possible that there are other unknown training parameters that could serve as mutation rules. To mitigate this potential threat, we deliberately chose model mutation rules that could directly or indirectly impact node interdependence in the prediction process.

*Threats to External Validity.* The external threats to validity mainly stem from the selection of the graph datasets as well as the GNN models adopted for our study. This threat is mitigated by the diversity of the subjects, as well as by the fact that we consider assessing not only natural inputs but also adversarial inputs.

*Threats to Construct Validity.* Our mutation rules are similar to the attacks used under graph adversarial settings. This may, in theory, create a bias in the experimental results related to adversarial test input prioritization. However, this threat is mitigated by two elements: first, we also apply NodeRank on natural inputs; second, the objective of the mutation is eventually to generate features for learning to rank initial inputs, not generating new samples that will be part of the test suite.

## VII. Related Work

### A. Test Prioritization Techniques

Test prioritization focuses on finding the ideal ordering of tests to detect more bugs in a limited time budget. In traditional software testing, a variety of approaches [14], [75], [76], [77], [78], [79] has been proposed. Mutation analysis has also been explored for test prioritization: Shin et al. [51] use a diversity-aware mutation adequacy criterion and demonstrate its effectiveness on large-scale developer-written test cases. Papadakis et al. [80] proposed mutating Combinatorial Interaction Testing models for test prioritization. Gökçe et al. [81] introduced a prioritized testing approach aimed at enhancing the testing capacity of ESG-based testing algorithms. ESG-based algorithms, as discussed by Belli et al. [82], focus on generating software test suites that meet specific criteria related to both coverage and execution cost. Gökçe et al.'s approach leverages adaptive competitive learning algorithms for training the neural networks utilized in this process. The core objective of their work is to improve the test capacity of existing algorithms by prioritizing the testing process. GÖKÇE et al. [83] introduced a model-based approach to test prioritization. Their method focuses on providing an effective algorithm for ordering test cases based on the perceived degree of preference by the tester. Unlike code-based approaches, which rely on prior knowledge such as fault counts, or source code, GÖKÇE et al.'s approach is radically different. It does not require prior knowledge about the system under test (SUT), making it suitable for a wide range of testing scenarios.

For DNNs, Feng et al. [10] have proposed DeepGini, which prioritized test inputs based on model uncertainty: a test input is more likely to be incorrectly predicted if the DNN model outputs similar probabilities for different classes. PRIMA [11] is currently the state-of-the-art DNN test prioritization approach. It is based on intelligent mutation analysis guided by learning-to-rank. NodeRank shares similarities with PRIMA in the use of mutation analysis. Unfortunately, PRIMA's mutation rules are not applicable to GNNs and their inputs. Our work is thus the first approach that specifically leveraged mutation testing adapted to GNNs in order to achieve test input prioritization.

## *B. Mutation Testing*

Mutation testing is commonplace in traditional software engineering [51], [80], where it constitutes a widely validated way to assess the quality of test cases. Mutation rules for traditional software have therefore been iteratively refined in the community. Recent studies have extended the applicability of mutation testing to various domains by focusing on adapting new mutation rules. Beyond simple bugs, Loise et al. [23] proposed 15 security-aware mutant operators to improve security testing. Beyond plain Java code, Deng et al. [84], [85] proposed novel mutant operators that are specifically designed to test Android applications (e.g., with event handling and activity lifecycle mutant operators).

Furthermore, in addition to the context of traditional software, several studies have investigated the application of mutation testing to DNNs and have proposed different mutation operators and frameworks. For instance, Ma et al. [86] proposed DeepMutation, a method to assess the quality of test data for DL systems using mutation testing. To achieve this, they designed a collection of source-level and model-level mutation operators to inject faults into the training data, programs, and DL models. The effectiveness of the test data is evaluated by analyzing the extent to which the injected faults can be detected. Later, Hu et al. [87] extended their work into a mutation testing tool for DL systems named DeepMutation++. This tool introduced new mutation operators for feed-forward neural networks (FNNs) and Recurrent Neural Networks (RNNs) and enabled the mutation of run-time states of an RNN. Another notable contribution is DeepCrime [21], a mutation testing tool that implements a set of DL mutation operators based on real DL faults. Shen et al. [88] proposed MuNN, a mutation analysis method for neural networks. MuNN defined five mutation operators based on the characteristics of neural networks.

## *C. Deep Neural Network Testing*

In order to improve the test efficiency of DNNs, existing studies [9], [10], [11], [87], [89], [90], [91], [92] has proposed several approaches to optimize the test process, which is mainly divided into two categories. The first one is test input prioritization, which has been elaborated in the above section. The second one is test selection, which focuses on selecting a small group of test inputs to precisely estimate the accuracy of the whole testing set to reduce labelling costs. Li et al. [9] proposed Cross Entropy-based Sampling (CES) to select representative test inputs for DNN accuracy estimation, which minimizes the cross-entropy between the selected set and the entire test set to ensure the distribution of the selected test set similar to the original test set. Chen et al. [89] proposed PACE for test selection and accuracy estimation. Pace clusters all the inputs in a test set into different groups and leverages the MMD-critic algorithm [93] to select prototypes from each group. In addition to improving DNN testing efficiency, existing studies [15], [86], [94], [95], [96] have also focused on measuring DNN testing adequacy. Pei et al. [95] proposed neuron coverage to assess the extent to which a test set covers the DNN model logic. Ma et al. [96] proposed DeepGauge, a set of coverage-based metrics that consider neuron coverage a good indicator to evaluate the adequacy of test inputs. Kim et al. [15] proposed surprise adequacy, which assesses the adequacy of test inputs by measuring their surprise with respect to the training set.

## VIII. Conclusion

To relieve the labelling-cost problem and improve the efficiency of GNN testing, we propose a novel test prioritization approach, NodeRank, which prioritizes test inputs that are more likely to be misclassified by the evaluated GNN model. NodeRank filled a gap in the literature: prioritization approaches that achieve state-of-the-art performance on DNNs are not suitable for GNNs since they ignore the interdependence between test inputs in graph-structured datasets. NodeRank leverages the concepts of mutation testing to perform test prioritization, with the aim of reducing the labelling cost in the process of evaluating a GNN model. Overall, NodeRank is a test prioritization approach that is model-based, input-based, and mutation testing-based. It utilizes mutation operations on both GNN models and test inputs to generate mutation features for each test input, facilitating test prioritization. The core idea is that: If a test input (node) can kill many mutated models and produce different prediction results with many mutated inputs, this input is considered more likely to be misclassified by the GNN model and should be prioritized higher. The specific process of NodeRank consists of two core steps: (1) NodeRank introduced three types of mutation rules to generate mutants from the perspective of the graph structure, node features, and the GNN model, respectively. (2) After obtaining the mutation results, NodeRank generated mutation feature vectors and utilized ensemble ranking models for test prioritization. Experimental results on 124 diverse subjects, considering natural and adversarial inputs, demonstrated the effectiveness of NodeRank. More specifically, NodeRank outperformed all the compared test prioritization approaches with an average improvement between 4.41% and 62.15%. Moreover, ablation experiments are performed to check that the different types of mutation features are all useful for the effectiveness of NodeRank.

## Data Availability Statement

Our replication package is available at https://zenodo.org/records/10049979.

## References

[1] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 6861–6871.

[2] M. Jiang et al., "Drug–target affinity prediction using graph neural network and contact maps," *RSC Adv.*, vol. 10, no. 35, pp. 20701–20712, 2020.

[3] P. Bongini, M. Bianchini, and F. Scarselli, "Molecular generative graph neural networks for drug discovery," *Neurocomputing*, vol. 450, pp. 242–252, 2021.

[4] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "GraphAF: A flow-based autoregressive model for molecular graph generation," 2020, *arXiv:2001.09382*.

[5] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–37, 2022.

[6] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 974–983.

[7] W. Fan et al., "Graph neural networks for social recommendation," in *Proc. World Wide Web Conf.*, 2019, pp. 417–426.

[8] C. Li, J. Ma, X. Guo, and Q. Mei, "DeepCas: An end-to-end predictor of information cascades," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 577–586.

[9] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting operational DNN testing efficiency through conditioning," in *Proc. 27th ACM Joint Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 499–509.

[10] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "DeepGini: Prioritizing massive tests to enhance the robustness of deep neural networks," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2020, pp. 177–188.

[11] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 397–409.

[12] L. Zhang, X. Sun, Y. Li, and Z. Zhang, "A noise-sensitivity-analysis-based test prioritization technique for deep neural networks," 2019, *arXiv:1901.00054*.

[13] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Input prioritization for testing neural networks," in *Proc. IEEE Int. Conf. Artif. Intell. Test. (AITest)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 63–70.

[14] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verification Rel.*, vol. 22, no. 2, pp. 67–120, 2012.

[15] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 1039–1049.

[16] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *ACM Trans. Softw. Eng. Method.*, vol. 30, no. 2, pp. 1–22, 2021.

[17] X. Dang, Y. Li, M. Papadakis, J. Klein, T. F. Bissyandé, and Y. L. Traon, "GraphPrior: Mutation-based test input prioritization for graph neural networks," *ACM Trans. Softw. Eng. Method.*, vol. 33, no. 1, pp. 1–40, 2023.

[18] I. D. Mienye and Y. Sun, "A survey of ensemble learning: Concepts, algorithms, applications, and prospects," *IEEE Access*, vol. 10, pp. 99129–99149, 2022.

[19] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdiscip. Rev., Data Mining Knowl. Discovery*, vol. 8, no. 4, 2018, Art. no. e1249.

[20] R. Polikar, "Ensemble learning," in *Ensemble Machine Learning*. Berlin, Germany: Springer-Verlag, 2012, pp. 1–34.

[21] N. Humbatova, G. Jahangirova, and P. Tonella, "DeepCrime: Mutation testing of deep learning systems based on real faults," in *Proc. 30th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2021, pp. 67–78.

[22] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep./Oct. 2011.

[23] T. Loise, X. Devroey, G. Perrouin, M. Papadakis, and P. Heymans, "Towards security-aware mutation testing," in *Proc. IEEE Int. Conf. Softw. Test. Verification Validation Workshops (ICSTW)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 97–102.

[24] Z.-H. Zhou and Z.-H. Zhou, "Ensemble learning," in *Machine Learning*, Singapore: Springer, 2021, pp. 181–210.

[25] A. Mohammed and R. Kora, "An effective ensemble deep learning framework for text classification," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 10, pp. 8825–8837, 2022.

[26] P. Goel, R. Jain, A. Nayyar, S. Singhal, and M. Srivastava, "Sarcasm detection using deep learning and ensemble learning," *Multimedia Tools Appl.*, vol. 81, no. 30, pp. 43229–43252, 2022.

[27] D. Che, Q. Liu, K. Rasheed, and X. Tao, "Decision tree and ensemble learning algorithms with their applications in bioinformatics," in *Software Tools and Algorithms for Biological Systems*, New York, NY, USA: Springer, 2011, pp. 191–199.

[28] J. Chen, Z. Li, and S. Qin, "Ensemble learning for assessing degree of humor," in *Proc. Int. Conf. Big Data Inf. Comput. Netw. (BDICN)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 492–498.

[29] F. Divina, A. Gilson, F. Goméz-Vela, M. García Torres, and J. F. Torres, "Stacking ensemble learning for short-term electricity consumption forecasting," *Energies*, vol. 11, no. 4, 2018, Art. no. 949.

[30] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, no. 2, pp. 212–261, 1994.

[31] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," 2019, *arXiv:1902.08412*.

[32] K. Xu et al., "Topology attack and defense for graph neural networks: An optimization perspective," 2019, *arXiv:1906.04214*.

[33] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 695–704.

[34] Y. Li, W. Jin, H. Xu, and J. Tang, "DeepRobust: A pyTorch library for adversarial attacks and defenses," 2020, *arXiv:2005.06149*.

[35] Z. Liu, Y. Dou, P. S. Yu, Y. Deng, and H. Peng, "Alleviating the inconsistency problem of applying graph neural network to fraud detection," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 1569–1572.

[36] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2005, vol. 2, pp. 729–734.

[37] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[38] P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 10–48550.

[39] C. Sun, A. Shrivastava, C. Vondrick, R. Sukthankar, K. Murphy, and C. Schmid, "Relational action forecasting," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 273–283.

[40] W. Pian, Y. Wu, X. Qu, J. Cai, and Z. Kou, "Spatial-temporal dynamic graph attention networks for ride-hailing demand prediction," 2020, *arXiv:2006.05905*.

[41] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[42] P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[43] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 1025–1035.

[44] J. Du, S. Zhang, G. Wu, J. M. Moura, and S. Kar, "Topology adaptive graph convolutional networks," 2017, *arXiv:1710.10370*.

[45] T.-Y. Liu et al., "Learning to rank for information retrieval," *Found. Trends® Inf. Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[46] R. E. Wright, "Logistic regression," 1995. [Online]. Available: https://psycnet.apa.org/record/1995-97110-007

[47] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[48] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[49] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 3149–3157.

[50] Q. H. Nguyen et al., "Influence of data splitting on performance of machine learning models in prediction of shear strength of soil," *Math. Prob. Eng.*, vol. 2021, pp. 1–15, 2021.

[51] D. Shin, S. Yoo, M. Papadakis, and D.-H. Bae, "Empirical evaluation of mutation-based test case prioritization techniques," *Softw. Test. Verification Rel.*, vol. 29, nos. 1–2, 2019, Art. no. e1695.

[52] Y. Lou, D. Hao, and L. Zhang, "Mutation-based test-case prioritization in software evolution," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 46–57.

[53] R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen, "Ablation studies to uncover structure of learned representations in artificial neural networks," in *Proc. Int. Conf. Artif. Intell. (ICAI)*, VA, USA, 2019, pp. 185–191.

[54] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: An analysis and survey," in *Advances in Computers*, vol. 112. Elsevier, 2019, pp. 275–378.

[55] M. Weiss and P. Tonella, "Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)," in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2022, pp. 139–150.

[56] Q. Hu et al., "Towards exploring the limitations of active learning: An empirical study," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 917–929.

[57] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002.

[58] S. Geisler, T. Schmidt, H. Sirin, D. Zügner, A. Bojchevski, and S. Günnemann, "Robustness of graph neural networks at scale," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, vol. 34, pp. 7637–7649.

[59] X. Zou et al., "TDGIA: Effective injection attacks on graph neural networks," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2021, pp. 2461–2471.

[60] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 2847–2856.

[61] J. Ma, S. Ding, and Q. Mei, "Towards more practical adversarial attacks on graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 4756–4766.

[62] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples on graph data: Deep insights into attack and defense," 2019, *arXiv:1903.01610*.

[63] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 40–48.

[64] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, 2020, pp. 1325–1334.

[65] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv:1706.06083*.

[66] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5149–5169, Sep. 2022.

[67] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Front. Comput. Sci.*, vol. 14, no. 2, pp. 241–258, 2020.

[68] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, vol. 32, pp. 8026–8037.

[69] D. Wang and Y. Shang, "A new active labeling method for deep learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Piscataway, NJ, USA: IEEE Press, 2014, pp. 112–119.

[70] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 1–10.

[71] P. E. McKnight and J. Najab, "Mann-Whitney U test," in *The Corsini Encyclopedia of Psychology*, Hoboken, NJ, USA: Wiley, 2010, pp. 1–1.

[72] G. Chryssolouris, M. Lee, and A. Ramsey, "Confidence interval prediction for neural network models," *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp. 229–232, Jan. 1996.

[73] K. Kelley and K. J. Preacher, "On effect size," *Psychol. Methods*, vol. 17, no. 2, pp. 137–152, 2012. [Online]. Available: https://psycnet.apa.org/record/2015-32022-018

[74] J. Cohen, "A power primer," 2016.

[75] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 733–752, Sep. 2006.

[76] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 523–534.

[77] J. Chen, Y. Bai, D. Hao, Y. Xiong, H. Zhang, and B. Xie, "Learning to prioritize test programs for compiler testing," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 700–711.

[78] J. Chen et al., "Coverage prediction for accelerating compiler testing," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 261–278, Feb. 2021.

[79] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based test case prioritisation: An industrial case study," in *Proc. IEEE 6th Int. Conf. Softw. Test. Verification Validation*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 302–311.

[80] M. Papadakis, C. Henard, and Y. L. Traon, "Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing," in *Proc. 7th IEEE Int. Conf. Softw. Test. Verification Validation (ICST)*, Cleveland, OH, USA. Liverpool, U.K. IEEE Comput. Soc. Press, 2014, pp. 1–10, doi: 10.1109/ICST.2014.11.

[81] N. Gökce, M. Eminov, and F. Belli, "Coverage-based, prioritized testing using neural network clustering," in *Proc. Comput. Inf. Sci., 21st Int. Symp.*, Berlin, Germany: Springer-Verlag, 2006, pp. 1060–1071.

[82] F. Belli and C. J. Budnik, "Test minimization for human-computer interaction," *Appl. Intell.*, vol. 26, pp. 161–174, 2007.

[83] N. Gökçe, F. Belli, M. Eminli, and B. T. Dincer, "Model-based test case prioritization using cluster analysis: A soft-computing approach," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 23, no. 3, pp. 623–640, 2015.

[84] L. Deng, J. Offutt, P. Ammann, and N. Mirzaei, "Mutation operators for testing android apps," *Inf. Softw. Technol.*, vol. 81, pp. 154–168, 2017.

[85] L. Deng, N. Mirzaei, P. Ammann, and J. Offutt, "Towards mutation analysis of android apps," in *Proc. IEEE 8th Int. Conf. Softw. Test. Verification Validation Workshops (ICSTW)*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 1–10.

[86] L. Ma et al., "Deepmutation: Mutation testing of deep learning systems," in *Proc. IEEE 29th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 100–111.

[87] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, and J. Zhao, "Deepmutation++: A mutation testing framework for deep learning systems," in *Proc. 34th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 1158–1161.

[88] W. Shen, J. Wan, and Z. Chen, "MuNN: Mutation analysis of neural networks," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 108–115.

[89] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient deep neural network testing," *ACM Trans. Softw. Eng. Method.*, vol. 29, no. 4, pp. 1–35, 2020.

[90] A. Guerriero, R. Pietrantuono, and S. Russo, "Operation is the hardest teacher: Estimating DNN accuracy looking for mispredictions," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 348–358.

[91] X. Dang, Y. Li, M. Papadakis, J. Klein, T. F. Bissyandé, and Y. Le Traon, "Test input prioritization for machine learning classifiers," *IEEE Trans. Softw. Eng.*, vol. 50, no. 3, pp. 413–442, Mar. 2024.

[92] Y. Li, X. Dang, L. Ma, J. Klein, Y. L. Traon, and T. F. Bissyandé, "Test input prioritization for 3D point clouds," *ACM Trans. Softw. Eng. Method.*, 2023.
[93] B. Kim, R. Khanna, and O. O. Koyejo, "Examples are not enough, learn to criticize! Criticism for interpretability," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, vol. 29, pp. 2288–2296.
[94] L. Ma et al., "DeepCT: Tomographic combinatorial testing for deep learning systems," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 614–618.
[95] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 1–18.
[96] L. Ma et al., "DeepGauge: Multi-granularity testing criteria for deep learning systems," in *Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng.*, 2018, pp. 120–131.

**Yinghua Li** received the master's degree from Chongqing University. He is currently working toward the Ph.D. degree with the University of Luxembourg. He was mainly engaged in the research of machine learning. His work mainly involves AI for SE and machine learning testing.

**Xueqi Dang** received the master's degree from King's College London. She is currently working toward the Ph.D. degree with the University of Luxembourg. Her work mainly involves graph neural network testing and machine learning testing.

**Weiguo Pian** received the bachelor's and master's degrees from Chongqing University, in 2018 and 2021, respectively. He is working toward the Ph.D. degree with the University of Luxembourg. His work mainly focuses on the intersection of machine learning and software engineering.

**Andrew Habib** received the Ph.D. degree from TU Darmstadt, Germany. He is a Research Scientist with the ABB Corporate Research Center, in Germany. Previously, he was a Postdoctoral Researcher with SnT, University of Luxembourg. His research interests include software engineering, machine learning, industrial AI, and their intersection.

**Jacques Klein** (Member, IEEE) is a Professor with SnT, University of Luxembourg. He coleads a group of about 25 researchers focusing on software security, software reliability, and intelligent software. He has standing experience and expertise in successfully running industrial projects; Android security, including both static analysis techniques for tracking privacy leaks and machine learning for identifying malware; and program repair. He published over 150 research papers in top journals/conferences.

**Tegawendé F. Bissyandé** is an Associate Professor with SnT, University of Luxembourg, where he leads a group of 25 researchers. He is an ERC Fellow and a Principal Investigator of several projects funded by the European Commission, the Fonds National de la Recherche, and by Industry partners. His research interests include software repair and software security with techniques based on program analysis and machine learning. He has published over 80 peer-reviewed research papers in various fields of computer science. As a native of Burkina Faso (West Africa), he is an enthusiastic Advocate of capacity building for higher education in Africa.