DeepState: Selecting Test Suites to Enhance the Robustness of Recurrent Neural Networks

Zixi Liu
zxliu@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yang Feng*
fengyang@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yining Yin
ynyin@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zhenyu Chen
zychen@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

ABSTRACT

Deep Neural Networks (DNN) have achieved tremendous success in various software applications. However, accompanied by outstanding effectiveness, DNN-driven software systems could also exhibit incorrect behaviors and result in some critical accidents and losses. The testing and optimization of DNN-driven software systems rely on a large number of labeled data that often require many human efforts, resulting in high test costs and low efficiency. Although plenty of coverage-based criteria have been proposed to assist in the data selection of convolutional neural networks, it is difficult to apply them on Recurrent Neural Network (RNN) models due to the difference between the working nature.

In this paper, we propose a test suite selection tool `DeepState` towards the particular neural network structures of RNN models for reducing the data labeling and computation cost. `DeepState` selects data based on a stateful perspective of RNN, which identifies the possibly misclassified test by capturing the state changes of neurons in RNN models. We further design a test selection method to enable testers to obtain a test suite with strong fault detection and model improvement capability from a large dataset. To evaluate `DeepState`, we conduct an extensive empirical study on popular datasets and prevalent RNN models containing image and text processing tasks. The experimental results demonstrate that `DeepState` outperforms existing coverage-based techniques in selecting tests regarding effectiveness and the inclusiveness of bug cases. Meanwhile, we observe that the selected data can improve the robustness of RNN models effectively.

---

*Yang Feng is the corresponding author.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

deep learning testing, deep neural networks, recurrent neural networks, test selection

1 INTRODUCTION

Deep Neural Networks (DNN) has been widely adopted in many fields to assist in solving various tasks, such as image classification [32], speech recognition [59], and natural language processing [7, 8, 13], etc. Although the DNN-driven systems have made remarkable progress in many aspects, they suffer from quality and reliability issues. The erroneous behaviors produced by DNN-driven systems could cause significant losses. For example, the RNN-driven dialogue system, Amazon's smart speaker Alexa could be fooled by some corner input cases as to makes creepy laughs, which scares the elderly and children and affects their lives [2]. The erroneous behaviors produced by DNN-driven systems could lead to misunderstanding, threats to personal safety [3], or even political conflicts [12]. Therefore, the testing and optimization of DNN-driven systems have become an urgent yet challenging task.

The DNN-driven systems are constructed upon the data-driven programming paradigm [29], which requires plenty of data with ground truth (i.e., labeled data) for model training and evaluation. Unfortunately, collecting a high-quality data set for building DNN-driven systems requires plenty of human efforts to label the data, which makes the process expensive and time-consuming. It is inefficient and impractical to employ the data collected from usage scenarios for the model's testing and optimization directly. Because in a massive data set, only a small amount of cases [34, 43, 56] which can trigger the system's potential errors are especially crucial for testing the DNN-driven systems. Furthermore, due to the nature of

DNN models, optimizing them via retraining with enormous data often costs plenty of time and computational resources.

Inspired by the effectiveness of code coverage in conventional software testing, researchers have proposed several neuron coverage (NC) criteria [23, 34, 35, 43] to evaluate the testing adequacy and guide test selection, thereby reducing the labeling cost and saving computational resources. Recently, the confidence-based Deep-Gini [16] and the feature-based PRIMA [57] have been proposed to prioritize tests. However, most of the existing neuron coverage criteria are mainly designed for the feedforward neural networks (FNN) [55], such as convolution neural networks (CNN) [6] and fully connected neural networks (FC) [44]. The general rationale of these criteria is to calculate the coverage rate by separately calculating the activation of each layer and neuron. However, it is difficult to transfer them to RNN which is a typical feedback neural network, due to the differences in working nature and network structures between RNN and FNN. Only a few test criteria [15, 24, 25] have been proposed for RNN models, which are calculated by analyzing the neuron states when the model is processing different test cases. The existing research often regards the hidden state at each iteration as the neuron layer in the FNN. Yet, such criteria are specially designed for detecting adversarial examples [63] that are manually generated with unrealistic transformations to attack the trained models. Moreover, the current research on neuron coverage is still in the controversial stage, and there are plenty of discussions on their effectiveness and usage scenarios [16, 20, 31, 60].

Different from these criteria, in this paper, we propose a test selection tool, namely DeepState, especially for identifying test cases that may trigger errors in the RNN-driven systems. DeepState is designed upon a stateful perspective rather than the neuron coverage of RNN. We measure RNN's uncertainty for a given test and analyze the output behaviors by expanding the hidden internal state of the RNN according to time steps. Specifically, we first capture the hidden state changes of RNN neurons over time steps. Then, we design two metrics, i.e., the changing rate and changing trend of the hidden states, to analyze the output behaviors of RNN models. To select test suites with a high bug detection rate from massive data, we design a test selection method based on the changing rate and changing trend of hidden states.

We implement DeepState and evaluate it on four RNN-based systems covering image classification and text classification tasks. Besides, we validate its effectiveness on a variety of RNN models, including the most commonly used LSTM, BiLSTM, and GRU. To evaluate the effectiveness of DeepState, we compared with random and existing neuron coverage-based selection methodologies. The experiment results show that DeepState can effectively select test cases with a high bug detection rate, which can help to test RNN models and find potential defects. Besides, we evaluate DeepState's effectiveness by calculating the inclusiveness of the selected test suites. The results under multiple selection ratios show that DeepState can filter out the tests that can reveal potential flaws in RNN models. Further, we evaluate its capability of improving the quality of RNN. We employ the selected data to retrain the original RNN model and record the accuracy improvements. The experiment results indicate that DeepState can improve the RNN models by retraining the model with the selected data.

In summary, the main contributions of this paper are as follows.

- **Approach.** We present an approach to model the internal states of RNN into a stateful perspective that can assist us in analyzing the behaviors of RNN models. Based on the stateful perspective, we further propose the metrics of the changing rate and changing trend to reflect the behaviors of RNN's hidden states.
- **Tool.** Based on the stateful perspective, we design and implement a test selection tool, namely DeepState, to assist in identifying a small portion of tests with high bug detection rate from a massive dataset efficiently. The source code of DeepState is publicly available [1].
- **Study.** We evaluate DeepState on the test selection of image and text classification. The experiment results demonstrate that the test suite selected by DeepState has a high bug detection rate and can be applied to retrain RNNs to improve the robustness of the models.

2 BACKGROUND

In this section, we introduce the preliminary knowledge of RNN, existing neural network test criteria, especially those for RNN models, and conventional test selection techniques.

2.1 Recurrent Neural Networks

The neural networks can be divided into feedforward neural networks and feedback neural networks based on their information propagation methods [49]. The feedforward neural networks [55], such as Convolutional Neural Network (CNN) [6] and Fully Connected neural networks (FC) [50], are composed of multiple layers of neurons. The neurons in each layer receive the output of the previous layer and output to the next. On the contrary, the neurons in the feedback neural network can not only receive signals from other neurons but also receive their own feedback signals.

The recurrent neural network (RNN) is a typical kind of feedback neural networks [55]. Fig. 1 illustrates a general structure of RNN. With the RNN unfolded, each neuron in RNN's hidden layer updates its state and weight iteratively over time steps [53]. The hidden states output h_t at time step t is calculated upon current input x_t as well as h_{t-1} from the previous time step, and then passed to the softmax layer to output the prediction y_t [64]. While RNN can output the prediction result at each time step, it can regard the output at the last time step as the final prediction of RNN. The LSTM (Long Short-Term Memory) model [21] is the most widely-used optimized RNN, which is primarily designed to solve the problem of gradient disappearance and gradient explosion in the training process of long sequences. The LSTM model adds a forget gate to the hidden layer to discard unimportant information, so it can better express the long-term and short-term dependence locally [36]. RNN is particularly effective in processing time-series data (i.e., a string of interrelated data) [37], such as image description, text generation, text classification, and other tasks.

2.2 Neuron Coverage Criteria

Neuron Coverage (NC) was first proposed by Pei et al. [43] to find inputs for DL systems that can trigger differential behaviors. Analogous to the source code coverage of conventional software
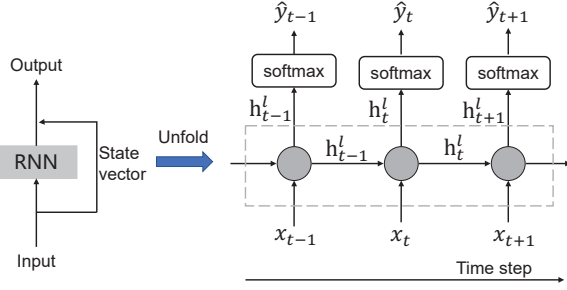
---

[1]https://github.com/SATE-Lab/DeepState

Figure 1: A general RNN structure.

testing, the neuron coverage is used to judge whether the output value of each neuron after passing the activation function exceeds a certain threshold k, and consider the neurons that exceed the threshold as activated, i.e., covered. The rate of $NC(k)$ for a test is defined as:

$$NC(k) = \frac{|covered\ neurons|}{|total\ neurons|}$$

Recently, some coverage criteria have been proposed especially for RNN, such as DeepStellar [15], testRNN [24, 25], and RNN-Test [19]. DeepStellar [15] calculates the basic state coverage (BSCov) and the basic transition coverage (BTCov) for quantitative analysis of RNNs. BSCov is designed to measure how thoroughly the test inputs T cover the major function region visited while training. The abstract states visited by the training inputs M and the test inputs T are denoted by $\hat{S_M}$ and $\hat{S_T}$, and the BSCov is defined as:

$$BSCov(T, M) = \frac{|\hat{S_T} \cup \hat{S_M}|}{|\hat{S_M}|}$$

BTCov targets the abstract transitions activated by various input sequences. BTCov compares the abstract transformation of the hidden state of the RNN model during training and testing phases, which are denoted as $\hat{\delta_T}$ and $\hat{\delta_M}$, respectively. Thus, the BTCov is defined as:

$$BTCov(T, M) = \frac{|\hat{\delta_T} \cup \hat{\delta_M}|}{|\hat{\delta_M}|}$$

testRNN [24, 25] proposed Step-wise Coverage (SC) for evaluating the changes in the output of the hidden layers in RNNs over the time step. The change of the hidden vector in adjacent time steps is defined as follows:

$$\Delta \xi_t^h = |\xi_t^{h,+} - \xi_{t-1}^{h,+}| + |\xi_t^{h,-} - \xi_{t-1}^{h,-}|$$

where $\Delta \xi_t^{h,+}$ represents the sum of all positive components in the hidden state h at time step t, and $\Delta \xi_t^{h,-}$ represents the sum of all negative components in the hidden state h at time step t. Then, SC is defined as:

$$SC = \frac{|\{\Delta \xi_t^h \geq v_{SC} | t \in \{1, \ldots, n\}|}{|\{\Delta \xi_t^h | t \in \{1, \ldots, n\}|}$$

In the formula, v_{SC} is a customizable threshold. In general, the maximum value that can be reached during training is selected as the threshold for testing.

RNN-Test [19] defines the hidden state coverage (HSCov) as the ratio of the hidden states that achieve the maximum value during testing. Assume H denotes all the hidden states of an RNN model of given inputs, which is a four-dimensional matrix of shape (T, L, B, E), where T, L, B, E are the number of time steps, layers, batches, and hidden units, respectively. The HSCov is defined as:

$$HSCov = \frac{|\{e | \forall e \in h, \forall h \in H, e = max(h)\}|}{|H|}$$

2.3 Test Data Selection

The test data selection technique is originally designed to select test cases in the regression testing scenario and save the execution time cost and resources. It selects the tests that are deemed necessary to verify the modified software from the existing test set. Suppose P is a procedure or program, P' is a modified version of P, and T is a set of tests (a test suite) created to test P. Tests that are valid for P may be redundant for P', because their execution trajectory does not go through the modified code in P'. The process of identifying the modified part of the execution trajectory through P' is called test selection.

Two primary coverage-based test cases prioritization techniques are known as the Coverage-Total Method (CTM) and the Coverage-Additional Method (CAM) [62].

(1) **Coverage-Total Method (CTM)**: it is regarded as the "next best" strategy. CTM takes no consideration on the selected tests. It always selects the test with the highest coverage from the candidate test suite.

(2) **Coverage-Additional Method (CAM)**: it is a selection strategy driven by the additional greedy algorithm. CAM dynamically adjusts the next selected test based on the selected tests. It always selects the test that can cover most uncovered code structures from the candidate test suite.

Inspired by the test selection in regression testing, in this paper, we propose `DeepState` to select a subset from massive data for RNN models.

3 APPROACH

We model the RNN internal state changes into a stateful view. Based on this stateful view, we design and implement a tool, namely `DeepState`, to select a subset of data from a large *unlabeled* dataset for the RNN model. As shown in Fig. 2, `DeepState` first unfolds the RNN model according to the time steps and captures the predicted label sequence based on a stateful view as discussed in Section 3.1. The label sequence can represent the internal state of the RNN hidden layers when predicting a given input data. Then, `DeepState` calculates the changing rate and the changing trend, which are presented in Section 3.2. These metrics are employed to evaluate the uncertainty and internal state transition of the RNN when processing the input data. We introduce the detailed implementation of `DeepState` selecting test cases in Section 3.3. Finally, Section 3.4 discusses how to enhance RNN quality with `DeepState`.

3.1 The Stateful View of RNN

RNNs are often employed to process the sequence data, i.e., a series of interdependent data streams, because the neurons in RNNs can dynamically update the hidden states based on the received information over time. Suppose an RNN is designed for processing a text classification task and the input is a sequence data $\mathbf{x} \in \mathcal{X}^N$, where $\mathcal{X}$ is the input domain, and N denotes the length of the input
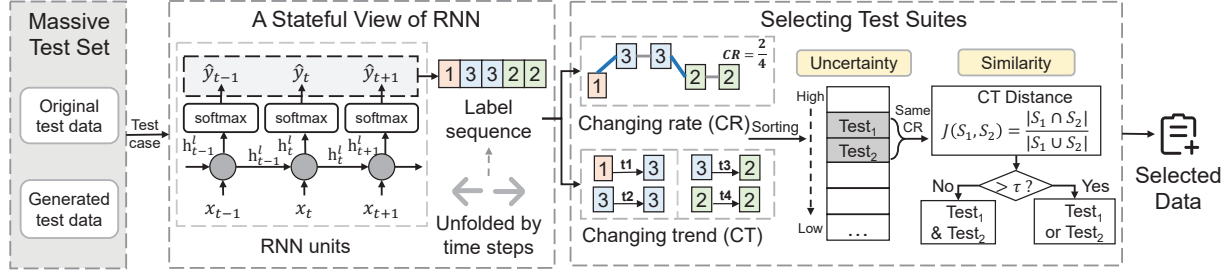
Figure 2: Overview of DeepState.

sequence. The i-th word in the input sentence $\mathbf{x}$ is represented as $x_i \in \mathcal{X}$. For a given input sentence $\mathbf{x}$, the RNN model iteratively receives each element in $\mathbf{x}$ (i.e., x_t is received at time step t) and maintains the international hidden state vector $\mathbf{s} \in \mathcal{S}^N$. Specifically, the time step represents the number of iterations and is equal to N. At time step t, the hidden state output is denoted as $s_t \in \mathcal{S}$, and s_t is updated upon both x_t and s_{t-1}. Then, the RNN model output the classification prediction result y_t through an activation function $f(\cdot)$. Generally, in multi-classification tasks, the Softmax function is applied to reflect the high-dimensional hidden state vector into the probability distribution vector for each label.

From the stateful view of RNN, we analyze the RNN model's prediction uncertainty and behavior features upon the hidden state at each time step. Because the high-dimensional hidden state vector $\mathbf{s}$ is difficult to analyze, we apply the corresponding predicted label y to reflect the information of the hidden state. For two adjacent time steps t and $t + 1$, the change of hidden state can be regarded as the corresponding label y_t and y_{t+1}. Note that in the earlier time steps, when the RNN model has not yet been well trained, it is natural for it to produce different prediction results and has low confidence for the output. As the received information increases, the RNN model's confidence for the prediction results also increases, and the prediction results at later time steps are supposed to be consistent.

Table 1: The example of hidden state outputs of two images in the MNIST dataset.

ID	Figure	Label sequence	Changes	Output
1	7	$[1, 1, 3, 0, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9]$	$1 \rightarrow 3;$ $3 \rightarrow 0;$ $0 \rightarrow 9$	9
2	7	$[1, 1, 2, 2, 2, 2, 2, 2, 3, 3]$	$1 \rightarrow 2;$ $2 \rightarrow 3$	3

To select tests that may trigger the potential erroneous behaviors of RNN models, we weigh the cases that cause a high changing rate of the RNN model's prediction over all time steps. For example, Table 1 shows a LSTM model's prediction process of two similar images from the MNIST [61] dataset. The correct label of these two images is nine, but the RNN mispredicted the second image (ID=2) to three, and the first picture (ID=1) is predicted correctly. Since the input data is a 28x28 image, the data is divided into 28 time steps (i.e., each time step inputs a row of pixels) and input into the

RNN model. We capture the hidden state output at each time step, and the corresponding predicted label. We list the prediction labels with confidence greater than 0.5 at each time step, which is shown in the label sequence column in Table 1. The label sequence for the first figure has 3 changes ($1 \rightarrow 3$, $3 \rightarrow 0$, $0 \rightarrow 9$) over the time steps. Similarly, the label sequence corresponding to the second figure has two changes ($1 \rightarrow 2$, $2 \rightarrow 3$). Although the changes' number of the second label sequence is one less than that of the first one, the first sequence is longer, and the second sequence has a change at the back of the sequence, which indicates the uncertainty of RNN.

3.2 Metric Computation

Based on the above stateful view, we employ the prediction label sequence to represent the hidden state output status of RNN processing a given input data. Then we propose two analysis metrics, i.e., the changing rate and changing trend of hidden states. The changing rate describes the uncertainty degree of the RNN model for a given input test case, while the changing trend describes the similarity of the output state of RNN processing different test cases.

Definition 3.1 (Label sequence). For a given input x, a label sequence is a list $Seq(x, c) = \{y_1, y_2, \ldots, y_N\}$, where c is the confidence threshold. At the i-time step, the RNN model's hidden state vector can be represented as the corresponding predicted label y_i through RNN's activation function. We maintain the predicted label with confidence greater than c in the label sequence. And N is the number of output labels with confidence greater than c in the output of all time steps, i.e., $|N| \leq |time\ steps|$.

Based on the label sequence, we measure the uncertainty of the RNN for a given input by calculating the changing rate of the predicted labels.

Definition 3.2 (Changing rate). The changing rate indicates the ratio of the number of adjacent labels that are not the same as the total sequence length in a label sequence $Seq(x, c)$.

The changing rate $CR(Seq(x, c))$ can be defined as:

$$CR(Seq(x, c)) = \frac{Count(y_{t+1} \neq y_t)}{|N - 1|} \tag{1}$$

Because the predictions at each time step may not have a high degree of confidence, especially in the first period of time, we calculate the changing rate on the label sequence, where each label is predicted with a confidence higher than a given threshold.

We calculate the weighted changing rate, considering that as the time step increases, RNN can receive more information, and the basis of the prediction result is sufficient. Thus, the later changes

in the $Seq(x, c)$ should be more weighted. Therefore, the changing rate we applied to select tests is defined as follows:

$$CR(Seq(x, c)) = \frac{Count(y_{t+1} \neq y_t) * Weight(t)}{\Sigma Weight(t)} \quad (2a)$$

$$Weight(t) = t^2, \text{ where } t = \{1, 2, \dots, N\}. \quad (2b)$$

We calculate the changing trend of the internal state of the RNN by mapping the specific transition between two states to the predicted label's transition. Specifically, the changing trend can be expressed as a set of conversions CT of two adjacent labels (y_t, y_{t+1}) in the label sequence $Seq(x, c)$.

Definition 3.3 (Changing trend). For a label sequence $Seq(x, c)$, the changing trend is a set $CT(Seq(x, c))$ of tuples composed of adjacent labels (y_{t-1}, y_t) in chronological order.

For example, the label sequence corresponding to the test ID=1 in Table 1 is $[1, 1, 3, 0, 0, 0, 9, 9, 9, 9, 9, 9, 9, 9]$, and the corresponding changing trend is $\{(1, 1), (1, 3), (3, 0), (0, 0), (0, 9), (9, 9)\}$. Similarly, the change sequence of the test ID=2 is $[1, 1, 2, 2, 2, 2, 2, 2, 3, 3]$, and the changing trend is $\{(1, 1), (1, 2), (2, 2), (2, 3), (3, 3)\}$.

To evaluate the similarity of two changing trends, we apply the Jaccard similarity coefficient [39] for evaluation. The similarity is calculated as follows:

$$J(CT_1, CT_2) = \frac{|CT_1 \cap CT_2|}{|CT_1 \cup CT_2|} \quad (3)$$

where CT_1 and CT_2 represent the changing trend of the RNN corresponding to two different test cases. The value range of $J(CT_1, CT_2)$ is $[0, 1]$. When $J(CT_1, CT_2)$ is greater than a given threshold, it indicates these two tests are similar to each other, and then we only keep one of them in the selection.

3.3 RNN Test Suite Selection Algorithm

DeepState weights the test cases with a high changing rate of label sequence, especially the predicted label changes at a later time step. Specifically, DeepState first sorts each test case in the dataset according to the changing rate, which indicates the uncertainty degree of RNN for a given test. We believe the uncertainty degree indicates the probability of the RNN model triggering bugs. Yet, it is not suitable to select many tests with the same changing rate directly because the selected test suites may not be representative. Thus, DeepState selects tests with different states' changing trends as the representative from the tests with the same changing rate.

Specifically, Algorithm 1 presents the process of DeepState selecting test suites for testing and optimizing the RNN models. For a given RNN model M to be tested and a massive data set D, DeepState first makes the RNN execute all test data in turn and calculates the changing rate CR and changing trend CT corresponding to each data (Line 1-7). Second, DeepState sorts all tests in reverse order according to the changing rate (Line 8) and sets a similarity threshold τ for the changing trend (Line 9). Third, for the test case set in D sorted by changing rate, DeepState traverses each test case in turn (Line 10). If the changing rates of the two test cases are different, both of them are selected (Line 11-13). For tests with the same changing rate, we calculate the changing trend similarity between them (Line 14-17). Both tests are selected if the distance is smaller than the threshold τ (Line 18-20). Otherwise, only one test

is selected. Finally, we check the last test $d_n \in D$. If the changing rate CR is different from that of d_{n-1}, and it is not selected, then we add it to $DataSelected$ (Line 26-28).

Algorithm 1: Selecting test suites

Input: $D = [d_0, d_1, \dots, d_n]$: The original data set
Input: M: The tested RNN model
Output: $DataSelected$: The selected data

1   $i = 0$;
2   $j = 0$;
3   **while** $i < n + 1$ **do**
4     $d_i.CR$ = getChangeRate(d_i, M);
5     $d_i.CT$ = getChangeTrend(d_i, M);
6     $i = i + 1$;
7   **end**
8   D = reverse_sortBy($D.CR$);
9   τ = setThreshold();   // If the distance between two CTs is greater than τ, then we keep one of them.
10   **while** $j < n$ **do**
11     $DataSelected$.add(d_j);
12   **if** $d_{j+1}.CR \neq d_j.CR$ **then**
13       $j = j + 1$;
14   **else**
15       $k = j + 1$;
16   **while** $d_k.CR = d_j.CR$ *and* $k < n + 1$ **do**
17   dis = JaccardDis($d_j.CT, d_k.CT$);
18   **if** $dis < \tau$ **then**
19           $DataSelected$.add(d_k);
20   **end**
21         $k = k + 1$;
22   **end**
23       $j = k$;
24   **end**
25   **end**
26   **if** $d_n.CR \neq d_{n-1}.CR$ *and* $d_n \notin DataSelected$ **then**
27     $DataSelected$.add(d_n);
28   **end**
29   **return** $DataSelected$

A running example. Assume that we have six tests A, B, C, D, E and F as well as an RNN with five time-steps. The RNN model is designed to classify tests into four classes, i.e., the label ranges from 1 to 4. Table 2 shows prediction label sequences, changing rates, and changing trends of some tests. Assume that the distance threshold for judging whether the changing trends of two tests are similar is 0.5, i.e., $\tau = 0.5$. These test cases are sorted into a descending order upon the value of the changing rate. DeepState first selects test case A because it has the highest changing rate, and no other case has the same changing rate value. Next, because the changing rate values of test cases B, C, and D are the same, DeepState first selects case B, and then calculate the Jaccard distance of changing trend between B and C, which is $1/7 (< 0.5)$. Because $1/7$ is less than τ, we expect that although these two test cases have the same changing rate, they trigger different states of the RNN, so DeepState selects both B and C. However, because the Jaccard distance of changing trend between C and D is $3/5 (> 0.5)$, which is greater than τ, DeepState passes D. Similarly, DeepState only selects one case between E and F because the Jaccard distance of changing trend between them is $3/5 (> 0.5)$.

In summary, for this example in Table 2, `DeepState` select test cases $\{A, B, C, E\}$ as its output. If the requirement is to select more than 4 test cases, then `DeepState` selects the test case D and F in turn. Because the internal state behaviors of RNN when processing tests C and D (or E and F) are very similar, there is no need to apply both tests for testing or optimization.

Table 2: An example to show how `DeepState` selects tests.

Test	Label Seq.	Changing rate	Changing trend
A	[1, 2, 3, 2, 1]	4/4	$\{(1, 2), (2, 3), (3, 2), (2, 1)\}$
B	[1, 2, 3, 3, 1]	3/4	$\{(1, 2), (2, 3), (3, 3), (3, 1)\}$
C	[2, 1, 3, 1, 1]	3/4	$\{(2, 1), (1, 3), (3, 1), (1, 1)\}$
D	[3, 1, 2, 1, 1]	3/4	$\{(3, 1), (1, 2), (2, 1), (1, 1)\}$
E	[2, 3, 3, 1, 1]	2/4	$\{(2, 3), (3, 3), (3, 1), (1, 1)\}$
F	[2, 3, 3, 3, 1]	2/4	$\{(2, 2), (2, 3), (3, 3), (3, 1)\}$

$$J(B, C) = \frac{|\{(3,1)\}|}{|\{(1,1),(1,2),(1,3),(2,1),(2,3),(3,3),(3,1)\}|} = \frac{1}{7}$$

$$J(C, D) = \frac{|\{(2,1),(3,1),(1,1)\}|}{|\{(1,1),(1,2),(1,3),(2,1),(3,1)\}|} = \frac{3}{5}$$

$$J(E, F) = \frac{|\{(2,3),(3,1),(3,3)\}|}{|\{(1,1),(2,2),(2,3),(3,1),(3,3)\}|} = \frac{3}{5}$$

3.4 Enhancing RNN with DeepState

To improve the practical performance of modern RNN-driven software applications, a general approach is to retrain the RNN model periodically with the data collected in the usage scenarios. The process often requires the data to be labeled properly; however, data labeling is an expensive and time-consuming task. Moreover, retraining the RNN models with a large amount of data is also a waste of time and resources. `DeepState` provides an automatic solution to this problem. It is designed for selecting a subset from a large *unlabeled* dataset. The selected data has the ability to discover potential defects of the RNN, which can activate the state transitions of different RNNs, ensuring the adequacy of the retraining data. `DeepState` allows us to find and label as many tests that may trigger RNN's erroneous behaviors as possible in a limited time budget. We observe that the tests selected by `DeepState` are more effective in enhancing RNN than the tests selected by coverage-based selection techniques.

4 EXPERIMENT DESIGN

We have conducted extensive experiments to evaluate the performance of `DeepState`. This section introduces the experiment settings and the research questions. To conduct the experiments, we implement `DeepState` upon Python 3.6.0 [4] and Keras [9] with TensorFlow 1.14.0 [5]. All experiments are performed on a Ubuntu 18.04.3 LTS server with Tesla V100-SXM2, one 10-core processor with 2.50GHz, and 32GB physical memory.

4.1 Datasets and RNN Models

As shown in Table 3, we experiment `DeepState` with image classification and text classification models. For each data type, we employ two widely-used datasets, and we applied each dataset on two RNN models to ensure the generality of the experiment results. Thus, we have eight combinations of dataset and model that cover popular RNN variants.

The MNIST [61] dataset is for handwritten digits recognition, containing 60, 000 training images and 10, 000 testing images. The Fashion [58] dataset is a dataset of Zalando's article images consisting of 60, 000 training images and 10, 000 testing images. Snips [11] natural language understanding benchmark is a dataset of over 16, 000 crowdsourced queries distributed among 7 different user intents. AgNews [1, 66] dataset is collected by more than 1 million news articles, which are divided into four classes, i.e., world, sports, business, and sci/tec. The total number of training samples is 120, 000, and the testing is 7, 600.

The LSTM (Long Short-Term Memory) [21] model is the most widely-used optimized RNN variant. It employs forget gates to record the long-term memory and short-term memory output of the RNN [41]. LSTM can effectively solve the problem of gradient disappearance and gradient explosion of the RNN model. The BiLSTM (Bidirectional Long Short-Term Memory) [17] is the bidirectional LSTM model, combining the forward LSTM and backward LSTM [41]. The GRU (Gated Recurrent Unit) [14] model is another variant of LSTM. It combines the forget gate and input gate of LSTM into a single gate, which is regarded as the reset gate [10].

Table 3: The details of Subject Datasets and RNN models.

Dataset	RNN Models	State Vec. Shape	# Trainable Parameters	Dataset Description
MNIST	LSTM	(128, 28)	81, 674	Handwritten numbers 0~9
	BiLSTM	(256, 28)	177, 866	
Fashion	LSTM	(128, 28)	69, 194	Zalando's article images (10 classes)
	GRU	(128, 28)	98, 186	
Snips	BiLSTM	(256, 16)	3, 194, 119	Intent recognition (7 classes)
	GRU	(128, 16)	2, 918, 279	
AgNews	LSTM	(128, 35)	214, 148	News classification (4 classes)
	BiLSTM	(256, 35)	427, 652	

4.2 Test Data Collection

To ensure that the experiment data can reflect the realistic data mutation in the usage scenarios, we employ widely-used benign data mutation operators rather than adversarial example generation techniques to augment test data. Specifically, we adjust the parameters of these operators and limit the mutation extent to be less than 5%. Thus, the augmented data maintains the same label as the original data. For each image classification model, we generate the data by adding perturbations with existing transformation techniques [38, 54], including contrast, brightness, shift, rotation, scaling, and shearing. All the implementation of image transformation is based on the Keras ImageDataGenerator tool [42]. For each figure in the original test set, we generate the corresponding figure with random parameters of multiple operators. For the text data, we generate the augmented data by adding perturbations with existing transformation techniques, including synonym replacement, back translation [51], word insertion [27], and abstract summarization [45], which is a data synthesis method for paragraphs. The textual transitions are implemented based on an open-source tool called nlpaug [33].

4.3 Baseline Approaches

We compare `DeepState` with other methods based on neuron coverage as introduced in Section 2.2. Then we sort the tests by applying CTM and CAM as introduced in Section 2.3 and then select the tests to satisfy the selection ratio requirement.

We apply both CAM and CTM strategies to sort the NC and SC (testRNN) and select the corresponding test cases. We apply HSCov (RNNTest) with the CAM selection strategy. HSCOV can only indicate which RNN hidden neuron is activated for each given input, so it cannot be selected after sorting by CTM. We apply BSCov and BTCov (DeepStellar) with the CTM selection strategy. The criteria proposed in DeepStellar can not be selected based on CAM because it converts the output state of the hidden layer into an abstract model and calculates the coverage. In addition, we also employ a random selection method as the baseline, i.e., randomly selecting a fixed proportion of test cases from the candidate set.

4.4 Research Questions

`DeepState` is designed for facilitating testers of RNN-driven systems to quickly select tests that can trigger the potential erroneous behaviors and effectively enhance the robustness. To this end, we empirically evaluate its performance based on the following three research questions (RQ).

RQ1. Effectiveness: Can the dataset selected by `DeepState` detect more defects than neuron-coverage-based methods?

Similar to the test case selection technology in traditional software testing, the goal of test case selection for deep learning models is also to filter out cases that can trigger potential defects. We provide answers to RQ1 by evaluating the bug detection rate of the test data set selected by `DeepState` and other baseline approaches.

To generate the candidate dataset, we apply the benign augmentation methods described in Section 4.2 on the original test set D_{test}. The augmented data is denoted as D'_{test}, which has the same size as D_{test}. We randomly select 30% size of the data from D_{test} and D'_{test}, respectively, and then merge them together to form a candidate test set D_{select}. Thus, The size of D_{select} is 60% of the size of D_{test}. To alleviate the potential bias, we follow the above step and repeat 30 times to randomly sample 30 candidate sets, denoted as $T_1, T_2, \ldots, T_{30}$. For each model and test suite, we apply all the selection techniques to select test suites from D_{select}, and we record the corresponding bug detection rate when the selection ratio $r = \{10\%, 20\%\}$. We employ T_i to denote the candidate set, and $T_{i,bug}$ to denote the bug cases in T_i, then the bug detection rate can be calculated as follows:

$$Bug\ Detection\ Rate(T_i) = \frac{|T_{i,bug}|}{|T_i|} \tag{4}$$

RQ2. Inclusiveness: Can `DeepState` filter out the bug test cases in the candidate test data set?

In RQ2, we investigate the extent to which `DeepState` selects tests that can reveal potential flaws in RNN models. Referring to the evaluation metric of inclusiveness proposed by Rothermel et al. [47], inclusiveness can measure the capability of regression test selection methods in choosing modification-revealing tests from the candidate set. Considering `DeepState` is designed to select a subset for manual labeling rather than for regression testing, we adapt the notion of inclusiveness to fit the performance evaluation of our

application scenario, which measures the capability of selecting bug-revealing tests from the candidate set. Specifically, we denote the original test set as D_{test}, the selected test set as D_{select}, and denote the tests that can detect erroneous behaviors in these two sets as $D_{test,bug}$, and $D_{select,bug}$, respectively. The inclusiveness can be calculated as follows:

$$Inclusiveness = \frac{|D_{select,bug}|}{|D_{test,bug}|} * 100\% \tag{5}$$

RQ3. Guidance: Can `DeepState` guide the retraining of an RNN to improve its accuracy?

Theoretically, the selected test suites with a high bug detection rate can be applied to optimize the RNN models. Thus, we propose RQ3 to evaluate whether the selected data can be applied for retraining and further enhance the robustness of RNN models.

To answer RQ3, we generate a candidate data set D_{select} containing augmented training data and the original training data. To keep D_{select} diverse, we randomly select 10% D_{train} as the dataset D_{part} and keep it separate from the original training process. Then, we generate the augmented data D'_{part} based on D_{part}. Finally, D_{select} is composed of D'_{part}, D_{part} and part of D_{train}, and the augmented data D'_{part} is kept to account for 25% of D_{select}.

We employ each selection technique to select 15% data from D_{select} to retrain the original model. To demonstrate that improvements in the accuracy are not influenced by inconsistencies in the data, we verify the model's accuracy improvement on both augmented test set D'_{test} and the mixed test set D_{mix} (augmented test data D'_{test} + original test data D_{test}) respectively. We consider that the robustness of an RNN model is improved when the accuracy of the retrained model on the original test set and the augmented test set both improved compared to the original model.

5 RESULT ANALYSIS

This section presents the experiment result and then analyzes the performance of our approach.

5.1 Answer to RQ1: Effectiveness

The average bug detection rate of 30 time experiments corresponding to 10% and 20% of selection rates is shown in Table 4. Fig. 3 shows the bug detection rate of each selection method when the selection ratio is set to 10%. Compared with baselines, `DeepState` has achieved the highest bug detection rate under different selection ratios, which indicates `DeepState` can effectively can help RNN models to detect more potential defects under given resource constraints. Among these four data sets, MNIST and Fashion are both image data, while Snips and AgNews are text data. Due to the short original text length and limited transformation methods, the text classification models have a relatively high accuracy score on the augmented text data set, so the bug detection rate is lower than that of the other two models.

Compared with random selection strategies, HSCOV (CAM), SC (CAM), NC (CAM), and `DeepState` can select tests with a higher bug detection rate, which indicates that these methods are more effective than random selection. Among them, the bug detection rate of the test cases selected by `DeepState` achieves the highest.
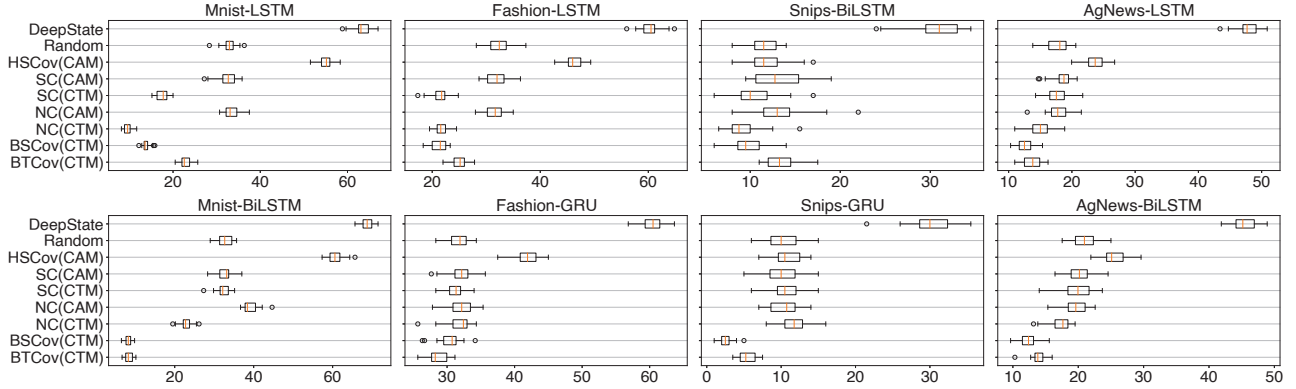
Figure 3: The bug detection rate of different selection methods with 10% selected tests.

This is because the CAM strategy can achieve the maximum coverage with a small number of tests, leading to some selected tests being invalid. Meanwhile, in fact, it is difficult to activate neurons at the earlier time step. Therefore, based on the selection strategy of CAM and coverage, only a small part of the tests with diversity can be selected. From the perspective of selection strategy, the CAM generally can select more bug cases than the CTM. This is because the CAM can determine the data to be selected next based on the selected results to ensure that the number of activated neurons is as large as possible. However, CTM directly sorts and selects according to the coverage rate. It is likely that a relatively similar test case is selected, which causes the bug detection rate to be relatively low. Therefore, compared with the random selection strategy, the selected test set based on CTM has a relatively lower bug detection rate, while the tests selected based on CAM generally achieve a higher bug detection rate.

Table 4: The bug detection rate of 10% and 20% selected tests.

	Model	Sel	Ran.	HSCov (CAM)	SC (CAM)	SC (CTM)	NC (CAM)	NC (CTM)	BSCov (CTM)	BTCov (CTM)	Deep State
MNIST	LSTM	10%	0.33	0.55	0.32	0.18	0.33	0.1	0.14	0.23	**0.63**
		20%	0.32	0.44	0.33	0.18	0.33	0.09	0.13	0.19	**0.58**
	BiLSTM	10%	0.33	0.61	0.33	0.32	0.39	0.23	0.08	0.08	**0.69**
		20%	0.33	0.49	0.33	0.32	0.36	0.23	0.1	0.1	**0.63**
Fashion	LSTM	10%	0.32	0.46	0.32	0.21	0.32	0.22	0.21	0.25	**0.6**
		20%	0.32	0.39	0.32	0.22	0.33	0.24	0.22	0.26	**0.55**
	GRU	10%	0.32	0.42	0.32	0.31	0.32	0.32	0.3	0.29	**0.61**
		20%	0.32	0.37	0.32	0.31	0.32	0.3	0.3	0.24	**0.56**
Snips	BiLSTM	10%	0.12	0.12	0.13	0.11	0.13	0.09	0.1	0.14	**0.31**
		20%	0.12	0.12	0.12	0.12	0.13	0.1	0.11	0.13	**0.27**
	GRU	10%	0.1	0.11	0.1	0.11	0.1	0.12	0.03	0.05	**0.3**
		20%	0.1	0.11	0.11	0.11	0.11	0.12	0.07	0.08	**0.25**
AgNews	LSTM	10%	0.18	0.24	0.18	0.18	0.18	0.15	0.13	0.14	**0.48**
		20%	0.18	0.22	0.18	0.18	0.19	0.16	0.12	0.13	**0.42**
	BiLSTM	10%	0.21	0.26	0.2	0.2	0.2	0.17	0.12	0.14	**0.46**
		20%	0.2	0.25	0.2	0.2	0.2	0.18	0.13	0.14	**0.41**

5.2 Answer to RQ2: Inclusiveness

For each model and data set, we evaluate the inclusiveness of the selected data set when the selection ratio $k = \{1\%, 2\%, 3\%, \ldots, 40\%\}$.

As shown in Fig. 4, the x-axis of each line chart represents the selection ratio k from the candidate data set, and the y-axis represents the inclusiveness of the selected data set. With the selection ratio k increasing, the inclusiveness of the test data selected by all methods is improved. Among them, DeepState outperforms other selection methods, which indicates that DeepState can select more bug test cases under the same select proportion.

In most combinations of models and data sets, the inclusiveness of the test suite selected by HSCOV (CAM) is better than that of random. We can safely conclude that HSCov is more effective than random as a selection strategy, but it is not as effective as DeepState. The effects of SC (CAM) and NC (CAM) are similar to those of random selection. This may be because the maximum coverage is reached quickly when applying the CAM select strategy, and the remaining part keeps the same effect as random selection. For the Snips dataset, it seems that the inclusiveness of all the methods except DeepState are similar to random selection. This may be due to the small size of the Snips data set and the short sentence length. Meanwhile, input sequence lengths in the Snips dataset are inconsistent, which may cause some methods that calculate coverage based on the output of the RNN hidden layer to be valid.

5.3 Answer to RQ3: Guidance

Table 5 shows the accuracy of all models after retraining with the data selected by all methods. Each grid represents the accuracy score of the model on the mixed test set after retraining. The data in parentheses represents the accuracy improvement on the test set compared to the original model. Additionally, we calculate the accuracy improvement after retraining with all the data in the candidate set as reference results. As can be seen from Table 5, in all models and data sets, compared to randomly selecting data for retraining, the data selected by DeepState can greatly improve the accuracy of the RNN models. In most combinations of models and data sets, the accuracy improvement of DeepState's selection method exceeds 50% of selecting all the data for retraining. We can safely conclude that the robustness of RNN can be greatly improved by retraining with a small amount of data selected through DeepState. The effect of DeepState on MNIST-BLSTM is slightly worse than HSCov (CAM). This is because the bidirectional neural network has more neurons than the other RNN models and requires more
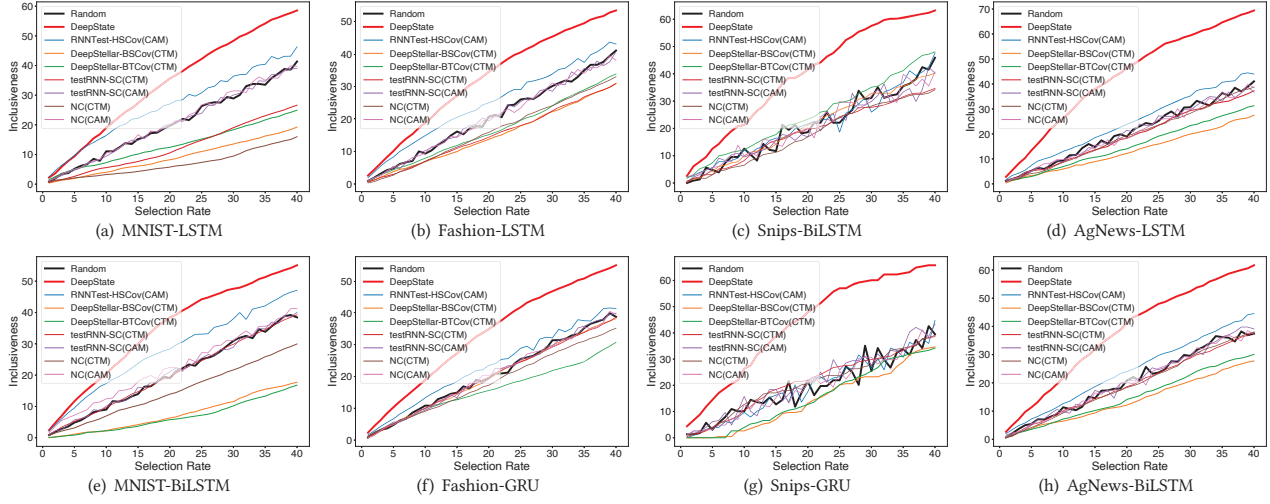
Figure 4: The inclusiveness of different selection methods.

test cases to maximize the HSCov coverage. The experiment results show that as the proportion of data selection increases, the effect of `DeepState` can gradually outperform the HSCov (CAM) selection method. Besides, although the retraining effect of SC (CTM) on Snips-BLSTM is slightly higher than that of `DeepState`, it is less effective than `DeepState` on other models, which indicates that the strategy of SC (CTM) is not stable and effective.

For the text classification tasks, the accuracy of the original model on the augmented test set is relatively high, and thus the accuracy improvement after retraining is also limited compared with the image classification models. The accuracy improvement of retraining with all candidate data may be less than selecting part of the data for retraining. This may be because there are an amount of original data in the candidate set, which leads to the model not learning the features of the augmented data well during retraining. In addition, the features of text augmentation are not as obvious as image transformation, which may affect the learning and optimization of RNN models.
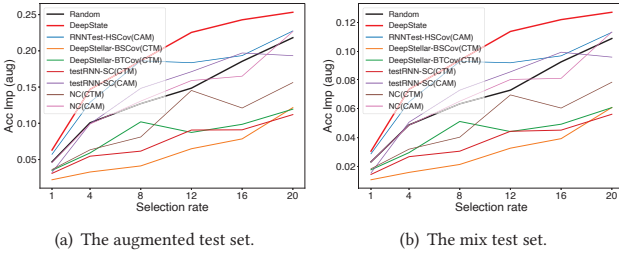

Figure 5: The MNIST-LSTM's accuracy improvement on augmented and mixed test set with different selection rate.

In our experiments, we evaluated the accuracy improvement after retraining under the selection ratio from 1% to 20%. Since most models have the same trend of accuracy after retraining, we show the accuracy of one model (MNIST-LSTM) under different selection

ratios, as depicted in Fig. 5. To demonstrate the improvements in the accuracy are not influenced by inconsistencies in the data, we evaluated the retrained model on both the augmented test set and the mix test set. The accuracy improvements on the augmented test set are greater than it on the mix test set. The accuracy improvements on the mixed test set have shown that the improvement of the retrained RNN model does not result from data inconsistency.

6 DISCUSSION

This section discusses the comparison with existing NC-guided test selection techniques, application scenarios of `DeepState`, and threats to validity.

6.1 Comparison with NC-guided Methods

By comparing `DeepState` with other existing test case selection methods based on coverage criterion, we demonstrate that `DeepState` is more effective than other existing methods. The selected tests have the capability of detecting bugs and can be employed to enhance the robustness of the RNN models. For a given test case, `DeepState` evaluates the uncertainty degree of the RNN model based on the changes of the hidden state's output over time steps. Compared with other coverage criteria that judge the neuron's activation based on the value of the hidden neuron's output, `DeepState` can better capture the state transition of RNN based on time steps. Further, different from the baseline methods applying CTM or CAM for test case selection, the selection algorithm of `DeepState` incorporates the advantage of both CTM and CAM. We evaluate the uncertainty of RNN for a given test case based on the changing rate, the process of sorting tests based on the changing rate is similar to CTM. Meanwhile, we select based on the similarity of their changing trends for cases with the same changing rate. This method can be regarded as CAM's selection strategy, which aims at selecting the cases that trigger different behaviors of the RNN. The selection strategy of `DeepState` makes the selected test set more diverse, and we conjecture this fundamentally benefits its performance.

Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen

Table 5: The RNNs' accuracy score on mix test data set after retraining with 15% selected tests.

Dataset	MNIST		Fashion		Snips		AgNews	
Method	LSTM	BLSTM	LSTM	GRU	BLSTM	GRU	LSTM	BLSTM
Random	76.25 (+9.86%)	74.76 (+7.84%)	75.26 (+4.29%)	75.20 (+3.75%)	90.15 (+2.22%)	89.25 (+0.06%)	80.05 (+0.17%)	81.66 (+0.28%)
HSCov (CAM)	74.78 (+8.39%)	**78.02 (+11.09%)**	75.73 (+4.77%)	75.08 (+3.62%)	88.99 (+1.06%)	89.45 (0.26%)	79.60 (-0.28%)	81.70 (+0.32%)
BSCov (CTM)	69.21 (+2.82%)	68.88 (+1.95%)	73.46 (+2.50%)	73.17 (+1.72%)	90.01 (+2.08%)	89.39 (+0.20%)	79.27 (-0.61%)	81.16 (-0.22%)
BTCov (CTM)	71.65 (+5.26%)	68.90 (+1.97%)	72.72 (+1.76%)	74.08 (+2.63%)	89.25 (+1.32%)	89.31 (+0.12%)	79.35 (-0.53%)	81.60 (+0.22%)
SC (CTM)	71.53 (+5.13%)	72.17 (+5.25%)	72.61 (+1.64%)	73.47 (+2.02%)	**90.39 (+2.46%)**	88.67 (-0.52%)	79.35 (-0.53%)	81.41 (+0.04%)
SC (CAM)	75.35 (+8.96%)	75.76 (+8.83%)	75.13 (+4.16%)	75.50 (+4.05%)	89.05 (+1.12%)	88.71 (-0.48%)	79.90 (+0.03%)	81.64 (+0.26%)
NC (CTM)	71.93 (+5.54%)	74.79 (+7.86%)	74.42 (+3.46%)	72.86 (+1.41%)	89.83 (+1.90%)	88.27 (-0.92%)	80.13 (+0.26%)	81.23 (-0.14%)
NC (CAM)	75.96 (+9.57%)	75.58 (+8.65%)	74.69 (+3.73%)	73.62 (+2.17%)	89.77 (+1.84%)	89.43 (+0.24%)	80.14 (+0.26%)	81.15 (-0.22%)
DeepState	**78.58 (+12.19%)**	77.40 (+10.47%)	**76.90 (+5.94%)**	**76.80 (+5.35%)**	90.03 (+2.10%)	**90.05 (+0.86%)**	**80.24 (+0.37%)**	**81.79 (+0.41%)**
100% tests	85.54 (+19.15%)	85.42 (+18.49%)	80.13 (+9.17%)	79.78 (+8.33%)	89.87 (+1.94%)	90.01 (+0.82%)	80.82 (+0.95%)	80.44 (+1.28%)

6.2 Application Scenarios

RNN is a kind of feedback neural network that specializes in processing time-series data. It is different from conventional feedforward neural networks, and many existing analysis criteria for traditional neural networks are not suitable for RNNs. DeepState can be applied to select tests from massive unlabeled data collected from the usage scenarios. It can automatically and efficiently identify tests that have a high probability of triggering the incorrect behaviors of RNN-driven systems from plenty of unlabelled data and thus help reduce the cost of manually labeling. Further, the experiment results also show that the accuracy of the model can be improved after retraining with the selected data. Applying small-scale selected data for retraining the model can also enhance the efficiency of model optimization and save computational resources. DeepState can be potentially applied to detect malicious attack samples against RNN models, such as backdoor attack data. It is capable of analyzing the states' changing trends of RNN after receiving different tests, because attack samples and ordinary samples are likely to cause significant changes in RNN models. We will explore the related study in our future work.

6.3 Threats to Validity

Test subject selection. The selection of datasets and RNN models selection is one of the primary threats to validity. There are many variants of the RNN model, including LSTM and multiple structures, and the corresponding effects can be different. On the other hand, the training of the RNNs relies on the dataset, and the quality of the data may have an influence on the model's accuracy. We alleviate this threat by employing four commonly used datasets, including image and text data types. Further, for each studied dataset, we employed two RNN models with different numbers of neurons and architecture to evaluate the performance of DeepState.

Parameters settings. Another threat could be the parameter settings in neuron coverages. To compare with other coverage-based test set selection methods, we reproduced other existing coverage methods for RNN, which may include parameters. With fine-tuning the parameters settings, the selected data could be different. To alleviate the potential bias, we follow the authors' suggested settings or employ the default settings of the original papers.

Test data simulation. The last threat to validity comes from the augmented test input generation. Although the augmented operators are common data noises in the actual environment, it is impossible to guarantee that the distribution of the real unseen input is the same as our simulation. To ensure the reliability of the augmented data, we refer to the existing image and text transformation methods based on some open-source tools. We believe that only some minor fine-tuning or adjustments are needed to supplement and simulate other transformations.

7 RELATED WORK

This section discusses the related work in two groups: (1) test selection methods for conventional software and (2) testing techniques for RNN models.

7.1 Test Selection

The test selection technique aims to find the test suite so that software testers can get the most benefits within limited resource budgets. The test selection technique was first proposed in conventional regression testing [18, 47, 48, 65], which aims at improving the testing efficiency of a modified software system.

Rothermel et al. [47] outlined the issues related to regression testing selection and proposed a series of metrics for evaluating regression testing selection techniques. Based on the relevant metrics, we made some fine-tuning on the inclusiveness to adapt for the testing of deep learning models. Rothermel et al. [48] proposed a safe test selection algorithm that constructs control flow graphs for the program and its modified versions. Then, the tests are selected based on these graphs and applied to execute the modified code from the original test suite. HyRTS [65] was the first hybrid regression test selection technique which performs analysis on multiple granularities to combine the advantages of traditional selection techniques with different granularities. Different from the approaches mentioned above, DeepState's test case selection technique is mainly aimed at testing RNNs, rather than traditional software testing based on data flow and control flow.

In the field of DNN testing, active learning [46], as a special case of machine learning, can interactively query the information source to label new data points for training DNN models [52]. The most commonly used query framework is uncertainty sampling [30], which aims to select unlabeled examples that the DNN finds hardest to classify. These uncertainty degree can be calculated based on conditional random fields [28], margin [26], and entropy [22]. Different from the active learning methods, DeepState is designed

to identify a subset of test data with the highest probability of revealing bugs in pre-trained RNN models. It is effective in testing and optimization scenarios, different from active learning focusing on the training process. Meanwhile, while most active learning techniques leverage the output of DNN models, such as boundary and posterior probability, as guidance for selecting data, `DeepState` employs structural characteristics of RNN models and captures their internal state to reach the goal.

7.2 Testing Recurrent Neural Networks

For the quality assurance tasks of RNN-driven software systems, some coverage-based testing approaches have been proposed in recent years. DeepStellar [15] adapts five coverage criteria of DeepGauge [34] to test and analyze RNN models, which is first transformed into a Discrete-Time Markov Chain (DTMC) [40] as an abstraction. TestRNN [24, 25] proposes a series of neuron coverage metrics of the LSTM network and develops a coverage-guided fuzzing approach for deep learning models with LSTM network structure. Then, some random mutation enhanced with the coverage is designed to generate test cases. RNN-Test [19] defines the hidden state coverage as the ratio of the hidden states that achieve the maximum value of all the hidden states during testing.

Different from these coverage-guided adversarial example generation methods, `DeepState` explores another solution to guide the test cases selection. `DeepState` analyzes the changing rate and changing trend of the hidden states in RNN models, and selects tests effectively among the unlabeled large-scale data sets. On the other hand, we focus on selecting test suites from massive data set for testing and optimizing the RNN models. Thus, the criterion of `DeepState` relies on the relationship among different cases rather than evaluating for a single test. Our study demonstrated that `DeepState` could more effectively select data set for RNN testing and optimization, and the oracle problem is alleviated.

8 CONCLUSION

In this paper, we propose `DeepState` for selecting massive test suites to enhance the robustness of RNNs. Based on a statistical view of RNN, we expand the output of the RNN model according to time steps and analyze the uncertainty degree of the RNN model as the time step changes. We design and implement a test suite selection tool `DeepState` by calculating the changing rate and changing trend of the RNN output sequence over time. The experimental results demonstrate that `DeepState` can effectively help testers to choose test cases with a high ability to detect bugs and improve the quality of the RNN models. `DeepState` can effectively ensure that the test data set has a high bug detection rate while greatly reducing the cost of data labeling and improving the efficiency of RNN model testing and optimization.

REFERENCES

[1] [n.d.]. AG's corpus of news articles. http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html. (Accessed on 08/21/2021).
[2] [n.d.]. Amazon promises fix for creepy Alexa laugh - BBC News. https://www.bbc.com/news/technology-43325230. (Accessed on 08/23/2021).
[3] [n.d.]. Amazon promises fix for creepy Alexa laugh - BBC News. https://www.bbc.com/news/technology-43325230. (Accessed on 08/29/2021).
[4] [n.d.]. Python Release Python 3.6.0 | Python.org. https://www.python.org/downloads/release/python-360/. (Accessed on 07/10/2021).
[5] [n.d.]. TensorFlow. https://www.tensorflow.org/. (Accessed on 08/23/2021).
[6] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. 2017. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*. 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186
[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[8] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2016. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038* (2016).
[9] François Chollet et al. 2015. Keras. https://keras.io.
[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[11] Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190* (2018).
[12] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. 2020. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review* (2020), 1–56. https://doi.org/10.1007/s10462-020-09866-x
[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).
[14] Rahul Dey and Fathi M. Salem. 2017. Gate-variants of Gated Recurrent Unit (GRU) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 1597–1600. https://doi.org/10.1109/MWSCAS.2017.8053243
[15] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 477–487. https://doi.org/10.1145/3338906.3338954
[16] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: Prioritizing Massive Tests to Enhance the Robustness of Deep Neural Networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 177–188. https://doi.org/10.1145/3395363.3397357
[17] A. Graves and J. Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 4. 2047–2052 vol. 4. https://doi.org/10.1109/IJCNN.2005.1556215
[18] Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, and Gregg Rothermel. 2001. An Empirical Study of Regression Test Selection Techniques. *ACM Trans. Softw. Eng. Methodol.* 10, 2 (April 2001), 184–208. https://doi.org/10.1145/367008.367020
[19] Jianmin Guo, Yue Zhao, Xueying Han, Yu Jiang, and Jiaguang Sun. 2019. Rnn-test: Adversarial testing framework for recurrent neural network systems. *arXiv preprint arXiv:1911.06155* (2019).
[20] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks?. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 851–862. https://doi.org/10.1145/3368089.3409754
[21] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
[22] Alex Holub, Pietro Perona, and Michael C Burl. 2008. Entropy-based active learning for object recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 1–8.
[23] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A Mutation Testing Framework for Deep Learning Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

1158–1161. https://doi.org/10.1109/ASE.2019.00126

[24] Wei Huang, Youcheng Sun, Xiaowei Huang, and James Sharp. 2019. testrnn: Coverage-guided testing on recurrent neural networks. *arXiv preprint arXiv:1906.08557* (2019).

[25] Wei Huang, Youcheng Sun, Xingyu Zhao, James Sharp, Wenjie Ruan, Jie Meng, and Xiaowei Huang. 2021. Coverage-Guided Testing for Recurrent Neural Networks. *IEEE Transactions on Reliability* (2021), 1–16. https://doi.org/10.1109/TR. 2021.3080664

[26] Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. 2009. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2372–2379.

[27] Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201* (2018).

[28] John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. (2001).

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[30] David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *SIGIR'94*. Springer, 3–12.

[31] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. Structural Coverage Criteria for Neural Networks Could Be Misleading. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results* (Montreal, Quebec, Canada) *(ICSE-NIER '19)*. IEEE Press, 89–92. https://doi.org/ 10.1109/ICSE-NIER.2019.00031

[32] D. Lu and Q. Weng. 2007. A Survey of Image Classification Methods and Techniques for Improving Classification Performance. *Int. J. Remote Sens.* 28, 5 (Jan. 2007), 823–870. https://doi.org/10.1080/01431160600746456

[33] Edward Ma. 2019. NLP Augmentation. https://github.com/makcedward/nlpaug.

[34] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 120–131. https://doi.org/10.1145/3238147.3238202

[35] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 100–111.

[36] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *Proceedings*, Vol. 89. Presses universitaires de Louvain, 89–94.

[37] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 5528–5531.

[38] Agnieszka Mikołajczyk and Michał Grochowski. 2018. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. 117–122. https://doi.org/10.1109/ IIPHDW.2018.8388338

[39] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. 2013. Using of Jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists*, Vol. 1. 380–384.

[40] James R Norris and John Robert Norris. 1998. *Markov chains*. Number 2. Cambridge university press.

[41] Christopher Olah. 2015. Understanding lstm networks. (2015).

[42] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. Keras Tuner. https://github.com/keras-team/keras-tuner.

[43] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2019. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Commun. ACM* 62, 11 (Oct. 2019), 137–145. https://doi.org/10.1145/3361566

[44] Philipp Petersen and Felix Voigtlaender. 2020. Equivalence of approximation by convolutional neural networks and fully-connected networks. *Proc. Amer. Math. Soc.* 148, 4 (2020), 1567–1581. https://doi.org/10.1090/proc/14789

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019).

[46] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. 2021. A survey of deep active learning. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–40.

[47] Gregg Rothermel and Mary Jean Harrold. 1996. Analyzing regression test selection techniques. *IEEE Transactions on software engineering* 22, 8 (1996), 529–551.

[48] Gregg Rothermel and Mary Jean Harrold. 1997. A Safe, Efficient Regression Test Selection Technique. *ACM Trans. Softw. Eng. Methodol.* 6, 2 (April 1997), 173–210. https://doi.org/10.1145/248233.248262

[49] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. 2015. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 4580–4584.

[50] Alexander G Schwing and Raquel Urtasun. 2015. Fully connected deep structured networks. *arXiv preprint arXiv:1503.02351* (2015).

[51] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709* (2015).

[52] Burr Settles. 2009. Active learning literature survey. (2009).

[53] Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404 (2020), 132306.

[54] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 1–48.

[55] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. 1997. Introduction to multilayer feed-forward neural networks. *Chemometrics and intelligent laboratory systems* 39, 1 (1997), 43–62.

[56] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 303–314. https://doi.org/10.1145/3180155.3180220

[57] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 397–409. https://doi.org/10.1109/ICSE43902.2021.00046

[58] Kashif Rasul & Han Xiao. 2017. Fashion. https://research.zalando.com/welcome/ mission/research-projects/fashion-mnist/.

[59] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256* (2016).

[60] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. 2020. Correlations between Deep Neural Network Model Coverage Criteria and Model Quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 775–787. https: //doi.org/10.1145/3368089.3409671

[61] Christopher J.C. Burges Yann LeCun, Corinna Cortes. 1998. MNIST. http://yann. lecun.com/exdb/mnist/.

[62] S. Yoo and M. Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *Softw. Test. Verif. Reliab.* 22, 2 (March 2012), 67–120. https://doi.org/10.1002/stv.430

[63] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems* 30, 9 (2019), 2805–2824. https://doi.org/10.1109/TNNLS. 2018.2886017

[64] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014).

[65] Lingming Zhang. 2018. Hybrid Regression Test Selection. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 199–209. https: //doi.org/10.1145/3180155.3180198

[66] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-Level Convolutional Networks for Text Classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 649–657.