RNN-Test: Towards Adversarial Testing for Recurrent Neural Network Systems

Jianmin Guo, Quan Zhang, Yue Zhao, Heyuan Shi, Yu Jiang and Jiaguang Sun

Abstract—While massive efforts have been investigated in adversarial testing of convolutional neural networks (CNN), testing for recurrent neural networks (RNN) is still limited and leaves threats for vast sequential application domains. In this paper, we propose an adversarial testing framework RNN-Test for RNN systems, focusing on sequence-to-sequence (seq2seq) tasks of widespread deployments, not only classification domains. First, we design a novel search methodology customized for RNN models by maximizing the inconsistency of RNN states against their inner dependencies to produce adversarial inputs. Next, we introduce two state-based coverage metrics according to the distinctive structure of RNNs to exercise more system behaviors. Finally, RNN-Test solves the joint optimization problem to maximize state inconsistency and state coverage, and crafts adversarial inputs for various tasks of different kinds of inputs.

For evaluations, we apply RNN-Test on four RNN models of common structures. On the tested models, the RNN-Test approach is demonstrated to be competitive in generating adversarial inputs, outperforming FGSM-based and DLFuzz-based methods to reduce the model performance more sharply with 2.78% to 37.94% higher success (or generation) rate. RNN-Test could also achieve 52.65% to 66.45% higher adversary rate than testRNN on MNIST LSTM model, as well as 53.76% to 58.02% more perplexity with 16% higher generation rate than DeepStellar on PTB language model. Compared with the traditional neuron coverage, the proposed state coverage metrics as guidance excel with 4.17% to 97.22% higher success (or generation) rate.

Index Terms—Adversarial testing, Recurrent neural networks, Coverage metrics

1 Introduction

As the core part of the current artificial intelligence applications, deep learning has made great breakthroughs in computer vision [1], natural language processing (NLP) [2], and automatic speech recognition (ASR) [3]. With the increasing deployments of deep neural network (DNN) systems in the safety- and security-critical domains, such as autonomous driving [4] and medical diagnose [5], ensuring the robustness of DNNs becomes an essential concern in both academic research and security communities.

However, it is demonstrated that state-of-the-art DNN systems [6] are easy to suffer attacks and produce completely wrong predictions, when fed with adversarial inputs which are nearly indistinguishable from original test inputs. This inspired numerous adversarial testing works devoted to generating adversarial inputs for DNNs, aiming to provide rich sources to train the DNNs to be more robust. The majority of these works [7], [8], [9] try to fool popular image classifiers by applying minute perturbations to the inputs. They exhibit high efficiency but achieve low testing completeness [10]. Recently, researchers [10], [11], [12] try to apply traditional software testing techniques over DNNs, with various neuron-based coverage criteria proposed to measure the testing completeness of DNN logics. These works could reach high testing coverage and produce numbers of adversarial inputs.

In spite of the efficiency of these works, they are largely limited to CNNs and image classification tasks. Overall, there are two main types of DNN, the convolutional neural networks (CNN) [13] and recurrent neural networks (RNN) [14]. They are of distinct structures and preferred for different kinds of tasks. CNN exhibits excellent competence in dealing with image processing tasks [15], [16], with thousands of neurons good at extracting image features. RNN is known for the iterative structure over cells and specific components dealing with context semantics, hence expert in handling tasks with sequential data, like natural language processing [17] and speech recognition [18]. Owing to the huge gap, the testing techniques and coverage metrics for the two types of DNNs are hard to fit the other.

So far, adversarial testing for RNN systems has received limited attention, especially those of sequence-to-sequence (seq2seq) tasks. Existing works [19], [20], [21], [22] concentrate more on sequence-to-one (seq2one) domains, performing well over classification tasks such as sentiment analysis [19], [20], [22], image classification [21], [22], and lipophilicity prediction [22], etc. But the large portion of seq2seq tasks leaves tested insufficiently, threatening their large-scale applications. Researchers [23], [24], [25] attack these models to perform abnormally in various manners, such as sampling words from blacklist [23] or producing attacker-targeted phrases [24]. Since without explicit class labels, there is no standard yet to recognize a generated sequence as the adversarial input avoiding false positives.

Moreover, existing coverage criteria [10], [11], [12] are mostly designed for CNNs and neurons, with a large gap to fit for RNNs. If taking the similar way of treating an RNN cell to be equivalent as a CNN neuron [26], an RNN model

- *J. Guo, Q. Zhang, H. Shi, Y. Jiang and J. Sun are with School of Software, Tsinghua University, Beijing National Research Center for Information Science and Technology, and Key Laboratory for Information System Security, Ministry of Education, Beijing, 100084, China.*
- *Y. Zhao is with Huawei Technologies Co., Ltd.*
- *H. Shi is corresponding author. Email: hey.shi@foxmail.com.*

Manuscript received Jan 6, 2021, revised Sep. 7, 2021.

is likely to compose a few neurons and reach full coverage in the early testing stage, resulting in few adversarial inputs discovered and lower testing capability, as presented in the later experiment section. Finally, recent works [21], [22] also proposed coverage guided testing methods with specific coverage criteria for RNN systems. They could generate lots of adversarial inputs for tested models (mainly of classification tasks), by mutating inputs directly (e.g. random noise) and employing coverage values as constraints to terminate testing. Despite their considerable efficiency, the specific inner behaviors of RNNs (e.g. state dependency) are not utilized in searching for perturbations and adversarial inputs.

Therefore, challenges for RNN testing are mainly summarized as threefold. First, adversarial testing methods for RNNs with seq2seq domains are rather inadequate, leaving threats for massive application scenarios. Second, neuron-based coverage metrics fail to consider characteristics of RNN structures and could not be adopted directly. Third, existing testing methods are limited in making use of distinct logics of RNN models.

Approach: In this paper, we propose an adversarial testing framework RNN-Test for RNN systems, especially those with sequential outputs. RNN-Test concentrates on the RNN structures and rids of remain parts for particular applications. According to the unique features of RNNs, we put forward a specific search methodology, which maximizes the inconsistency of RNN state dependencies to obtain adversarial inputs. Meanwhile, we also design two state-based coverage metrics for different RNN models to exercise more internal behaviors and guide to discover adversarial inputs in irregular space. They are then combined as a joint optimization problem, which is to maximize the state inconsistency and state coverage. Finally, it will be solved to acquire perturbations in a gradient-based way. When obtained the perturbations, adversarial inputs will be crafted by applying perturbations to original test inputs in different ways for various kinds of inputs. In the end, we adopt model performance metrics to identify the adversarial inputs and assess their qualities, which are generally available for RNN variants. In this way, RNN-Test provides a scalable and extensible solution for RNN testing.

Evaluation: We evaluate the RNN-Test approach over four RNN systems dealing with different tasks, including three seq2seq models [27], [28], [29] and one seq2one model [30]. The RNN-Test approach could efficiently acquire adversarial inputs of high quality, which reduce the model performance sharply while nearly imperceptible to original inputs. Compared with baselines which are two popular techniques (FGSM [7] and DLFuzz [31]) adapted here for RNN testing, our approach achieves more performance reduction with higher success (or generation) rate. Taking DeepSpeech ASR model as an example, RNN-Test could decline the model performance by 17.29% higher WER, 3.61% lower BLEU with 10% higher success rate, in contrast with the FGSM-based method. With respect to most relevant RNN testing works, RNN-Test could achieve 52.65% to 66.45% higher adversary rate than testRNN [22] on MNIST LSTM model [32], and magnify 53.76% to 58.02% more perplexity with 16% higher generation rate than DeepStellar [21] on PTB language model [27].

Furthermore, coverage guidance as pure optimization is first demonstrated with diverse searching capability for adversarial inputs compared with other methods. The proposed state coverage guidance achieved 4.17% to 97.22% higher success (or generation) rate than neuron coverage guidance, and even best performance on the spell checker model. Adversarial inputs obtained by RNN-Test are of high quality. Besides reducing the model performance sharply with minute perturbations and high time efficiency, they could also improve the model by retraining, such as 12.58% improvement of test perplexity on PTB language model.

Contribution: Our work has the following contributions:

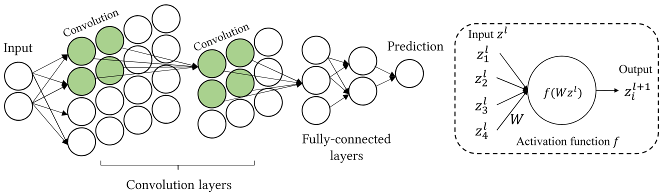
- We design a novel search methodology based on the inner logics of RNNs, which maximizes the inconsistency of RNN state dependencies to produce adversarial inputs efficiently.
- We propose two state-based coverage metrics customized for RNNs, mainly as guidance for adversarial testing. We demonstrate that coverage guidance has a diverse searching capability for adversarial inputs compared with other methods.
- We design and implement the adversarial testing framework RNN-Test. To our best knowledge, this is the first step towards systematically testing the seq2seq domains for RNN systems. It is effective and scalable for variants of RNNs, outperforming FGSM- and DLFuzz-based methods as well as testRNN (on MNIST LSTM model) and DeepStellar (on PTB language model).

2 Background

2.1 Deep Neural Network

In the following, we will describe the two main kinds of DNN, convolutional neural networks (CNN) [13] and recurrent neural networks (RNN) [14].

Convolutional neural network and neuron. Fig. 1a shows the simplified structure of a typical CNN. CNN keeps the fundamental feed-forward structure, where each neuron is connected with neurons of adjacent layers while no connections with those of the same layer. They are broadly used in image processing tasks [15], [16], with specific convolution layers good at extracting image features. Besides, CNN adopts classical DNN neurons, as shown in Fig. 1b. The neuron output is a single value transformed by a non-linear activation function, which is usually ReLU (Rectified Linear Unit).



(a) Typical CNN structure (b) Classical neuron

Fig. 1: CNN structure and neuron

Recurrent neural network and cell. Fig. 2 illustrates the basic structure of RNN, where an elementary RNN cell (noted as the square) iteratively makes predictions $\hat{y}$ based on inputs x and intermediate outputs h, which are referred to as hidden states. When it is unfolded, the input sequence x is fed to the RNN cell as a series of time steps, where x could be a sentence and x_t is the t-th word. Moreover, each prediction $\hat{y_t}$ could be the predicted word right after x_t based upon the received sequence.

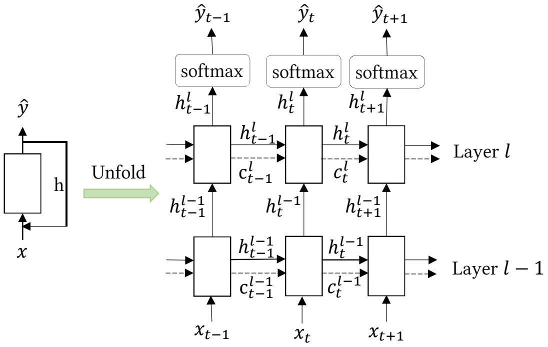
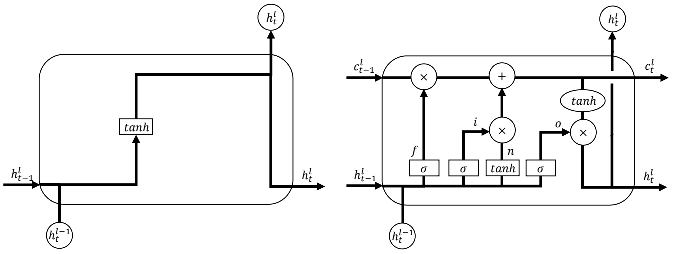


Fig. 2: RNN structure

In contrast to CNN neuron, the hidden state output h_t^l of the cell at time step t of layer l is decided by current input h_t^{l-1} from the previous layer as well as h_{t-1}^l from the prior step in the same layer, and then passed forward to compute the softmax predictions. Due to this key design, RNN excels in making use of the interior contextual semantics of sequential inputs.

Nevertheless, the basic RNN is unable to learn the semantic dependency within longer time steps. LSTM (Long short-term memory) [33], [34] and GRU (Gated recurrent unit) [35] networks are generally deployed solutions, bringing gate mechanisms to RNN cell. Fig. 3 provides the general RNN cell and LSTM cell as the example, in which f, i, n, o stand for various gates[1], and σ for sigmoid function. Unlike the plain RNN cell using a single tanh function to transfer the data, LSTM cell relies on cell states c (special of LSTM networks) to maintain the context, to which multiple gates could remove or add information, and finally to produce the hidden state outputs.



(a) General RNN cell (b) LSTM cell

Fig. 3: RNN cell

2.2 Limitations of existing coverage metrics

While numerous coverage metrics [10], [11], [12] are proposed for DNN testing, they are mostly based on CNN neurons and hard to satisfy RNN testing. First, an RNN usually comprises one or two layers each with several cells when unfolded, much fewer than a CNN usually of ten more layers each with hundreds of neurons. Second, sigmoid and tanh are conventionally used for an RNN cell whereas ReLU is the most choice for a CNN neuron. This is critical because that the value range of ReLU is $[0, \infty)$ while $[0, 1]$ for sigmoid and $[-1, 1]$ for tanh, leading to much narrowed value ranges of RNN states (hidden state value not exceeding ± 1) compared to those of CNN neurons (e.g. neuron values of a large model like VGG-16 [15] could be greater than 1000).

Unfortunately, neuron-based coverage metrics fail to consider these critical characteristics. We evaluate neuron coverage [10] on our tested models, which treats the hidden states of each RNN cell as the equivalent output of a CNN neuron. As for the PTB language model comprising two layers each with 10 time steps, there will be only 20 neurons in total. The neuron coverage reaches 100% with at most 4 inputs even taking a higher threshold 0.5. When for adversarial testing on this model, it fails to find any adversarial inputs, shown in Table 4 of § 5. Besides, the narrow value ranges of RNN states largely limit the application of popular coverage criteria [12], [36] in RNN testing. Since they measure over multi-section (e.g. 1000) neuron value ranges discriminated by training and testing inputs, but RNN states of training and testing sets are with similar thin ranges and hard to differentiate.

Recently, DeepStellar [21] abstracts RNN models as Discrete-Time Markov Chain (DTMC) models and then adapts coverage metrics of [12] for testing. Another work testRNN [22] designs novel coverage metrics for LSTM models to quantify temporal relations in RNNs. As to these coverage criteria, they primarily characterize abnormal values in testing, which are recognized by thresholds set according to training data. In this paper, we define coverage metrics trying to capture key features of RNN states with no aid of training data, where state statistics are extracted immediately during inference phase.

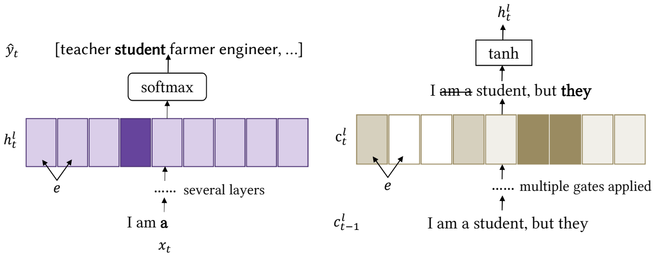
3 State Coverage Metrics

In this section, we propose two state coverage metrics based on unique features of RNNs. Hidden state coverage (HS_C) is designed to capture RNN prediction logics, which could be universally applied to RNN models. Due to the widespread deployments of LSTM networks and their peculiar design of cell states, cell state coverage (CS_C) is specially designed for LSTM models.

3.1 Hidden State Coverage

As discussed in § 2.2, RNN states cannot be regarded to be identical to CNN neurons. Fig. 4 provides a simple illustration of the inner logics of hidden states and cell states. In Fig. 4a, a hidden state h_t^l represents the output of each RNN cell, which is a vector containing hundreds or thousands of units. Here each rectangle represents each unit e, where a darker color means a higher value. When to predict the next word following "a", these hidden state units will be mapped to a list of candidates. If a hidden state unit is the maximum, its mapped candidate will probably be the prediction result.

[1] Gates of GRU are different but similar in design, not listed here.



(a) Hidden state

(b) Cell state

Fig. 4: Illustrations of RNN states.

Therefore, hidden state units of each vector h lead to varied prediction results for each step. During the testing phase, it is meaningful for each hidden state unit to be the maximum in the vector and perform predictions, especially those of the last layer fed to the softmax layer. To summarize, we define hidden state coverage as the ratio of such hidden state units of all the hidden state units during testing. Note that RNN networks usually compose one RNN layer or sometimes two layers, out of which a state unit of higher value in the first layer is more likely to correspond to higher values in the last layer. Based on the few layers and thorough state information needed, we leverage all the hidden state units to benefit the coverage guided testing. Formally, the definition is given in formula (1).

$$HS\_C = \frac{|\{e_0 \mid \forall h \in H, e_0 \in h, \forall e \in h, e_0 >= e\}|}{T \times L \times B \times E} \tag{1}$$

Assume H denotes all the hidden state units of an RNN model of given test inputs, which is a four-dimensional matrix of shape (T, L, B, E), where T, L, B are the number of the time steps[2], layers and batch, respectively. E is the number of units of a hidden state. Though H varies among RNN models, T, L, B, E are always the necessary components, where batch is to accelerate computations by feeding multiple inputs simultaneously. Thus, $T \times L \times B \times E$ will be the total number of hidden state units in matrix H. Here, a specific hidden state $h \in H$ contains E units and is denoted as its index of H, which is $[t, l, b]$, where $t \in \{1, 2, \dots, T\}, l \in \{1, 2, \dots, L\}, b \in \{1, 2, \dots, B\}$. That means, h is the output of the RNN cell of the t-th step in the l-th layer for the b-th input. For a state unit $e_0 \in h$, e_0 is covered if its value is the maximum out of any $e \in h$.

Note that hidden state coverage is designed of general-purpose for RNN models. As for various RNN variants including LSTM, Bi-LSTM (Bidirectional LSTM networks) and GRU models, hidden states work in the same way once obtained from RNN cells of distinct designs. Taking a Bi-LSTM model as an example, its forward and backward hidden states of one layer are concatenated together to feed to the next layer, and similarly to compute predictions.

3.2 Cell State Coverage

As in Fig. 3b, the cell states and gates are activated by functions sigmoid and tanh. The sigmoid function of three gates outputs values between 0 and 1, determining how much of each cell state to keep. The tanh function pushes the states between -1 and 1, for gate n to add information to cell states, and for cell states c to compute the hidden states. Thus, each cell state value protects the contexts. Fig. 4b illustrates the cell states, where c_t^l is the output of each LSTM cell which is also a vector of cell state units. Similarly, a rectangle is also a cell state unit e and a darker color for a higher value. When to predict the predicate after "they", c_t^l will receive contexts from c_{t-1}^l to keep key semantics and remove those invalid, and then to compute predictions.

In this paper, we design cell state coverage over different value ranges standing for degrees to keep contexts. In the experiments, cell state values mostly fall into the central range while few be the boundary value. We suppose that covering more of each section (5 sections in this paper), especially boundary sections, could explore more context space. The formal definition is given in formula (2).

$$CS\_C_{sec_i} = \frac{|\{e \mid \forall c \in C, \forall e \in c, \tanh(e) \in sec_i\}|}{T \times L \times B \times E} \tag{2}$$

Here, all the cell state units of an RNN model fed with given test inputs are denoted as C, which is also a matrix of shape (T, L, B, E). The value range of function tanh is split to Sec sections and each section is $sec_i = [v_{i-1}, v_i]$, where $-1 \leq v_i \leq 1$. For a specific cell state $c \in C$, it is also denoted by the index of C as $[t, l, b]$. If a cell state unit $e \in c$ and its activation value $\tanh(e) \in sec_i$, $i \in \{1, 2, ..., Sec\}$, then e is covered in sec_i.

Furthermore, these two coverage criteria only require the test set to extract coverage information without the training process. Thereby, we could measure how extensively the test inputs exercise RNN logics and benefit adversarial testing with state coverage metrics as guidance, without additional resources extracted from training data.

4 RNN-Test Design

In this section, we present a technical description of RNN-Test in detail. Fig. 5 depicts the overall workflow. Given the tested model, RNN-Test will focus on the RNN structures without other components for particular tasks. For original test input x, we first extract the hidden states and cell states of each RNN cell by state wrapping, without affecting its inherent process. These states are crucial for the subsequent state inconsistency guided search, maximizing state inconsistency and state coverage to generate adversarial inputs. Unlike the usual idea of increasing the model cost [7], [37] or the probabilities of targeted classes [9], [31], RNN-Test tries to increase the inconsistency of RNN states against their inner dependencies (shown in Fig. 5, the part of obj rounded with blue dashed frame violates data dependencies marked with blue lines in the model). In this way, RNN-Test could search for adversarial inputs in a lightweight and scalable means. Meanwhile, RNN-Test also tries to cover more states and exercise more system behaviors during testing, guided with specific state coverage information for different models. Then, the joint optimization problem will be solved in a gradient-based manner and acquire minute perturbations.

[2] For models could be fed with inputs of non-equal steps, T will be adjusted according to the length of each input.

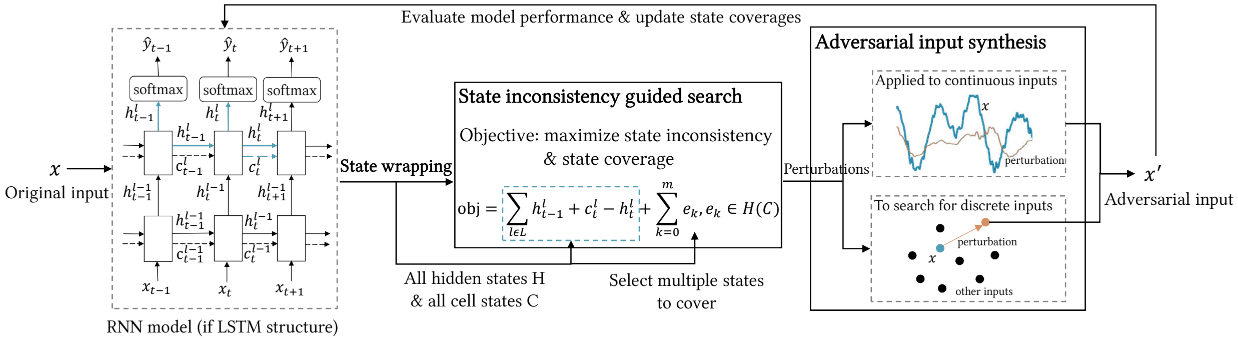

Evaluate model performance & update state coverages
softmax
Original input
RNN model (if LSTM structure)
State wrapping
State inconsistency guided search
Objective: maximize state inconsistency & state coverage
All hidden states H & all cell states C
Select multiple states to cover
Perturbations
Adversarial input synthesis
Applied to continuous inputs
perturbation
To search for discrete inputs
other inputs
Adversarial input

Fig. 5: Architecture of RNN-Test

Once obtained the perturbations, adversarial inputs are easy to acquire for models with continuous inputs like speech, by applying perturbations directly to original inputs. For models with discrete inputs like NLP tasks, the perturbation applied to the test input probably will not lead to a legal input. Here, we adopt the nearest one as the adversarial input after iteratively scaling the perturbation, thus avoiding the invalid input. Finally, these adversarial inputs will be assessed concerning the tested model for the performance and coverage, to improve subsequent testing efficiency. The detailed descriptions for the above steps are given below.

4.1 State wrapping

In the inherent implementation of an RNN model, there are two data structures accessible in the inference: all the hidden states of the last layer, and all the hidden states and cell states (if LSTM underlying) of the last time step. For RNN testing, exploiting all the states should be a better choice for thoroughly searching for adversarial inputs. Therefore, we wrap the RNN cell implementation and keep all the hidden states and cell states of RNN cells in every layer and time step. With straightforward configurations, state wrapping will not interfere inner computation of the tested models. Note that this step is not expensive and will not affect the time efficiency, with an open-source library (20 lines of Python code). It is based on fundamental "RNNCell", the parent class of various cell implementation, making it possible to be generalized to most RNN models.

4.2 State inconsistency guided search

The state inconsistency guided search is the core portion of RNN-Test, an optimization problem composed of two parts. It is formulated in equation (3), where the first part (obj_1) is referred to as adversary search, and the second part (obj_2) is as coverage guidance. Here, we treat the two parts as a single optimization problem jointly, intending to search for appropriate adversarial inputs with lower computation overhead. In addition, the two parts are free to be united together or alone, or even substituted with those of other methodologies, thus offering multiple possibilities of discovering adversarial inputs.

$$\begin{cases} obj = obj_1 + obj_2 \\ obj_1 = \sum_{l \in L} h_{t-1}^l + c_t^l - h_t^l \\ obj_2 = \sum_{k=0}^{m} e_k, \ e_k \in H(C) \end{cases} \tag{3}$$

Adversary search. As illustrated in Fig. 3 and Fig. 5, an RNN or LSTM cell receives concatenated previous outputs h_{t-1}^l and h_t^{l-1}, and then applies gate functions (sigmoid or tanh) which are monotonically increasing, to finally obtain its output h_t^l. Note that GRU cell is of the similar case. Consequently, hidden state vector h_t^l probably has a positive correlation with the inputs h_{t-1}^l, h_t^{l-1} and intermediate outputs c_t^l (if c implemented), which is demonstrated to be a basic rule (though not universal in few cases) in our validations.

Inspired by the state dependency, a novel methodology is designed to craft adversarial inputs specially for RNN models. Here, RNN-Test tries to increase h_{t-1}^l and c_t^l while decrease h_t^l simultaneously, which intentionally violates their inner dependencies to lead the model to exhibit unusual behaviors. Then the violated dependencies will spread across the whole model. In this manner, RNN-Test is able to search for adversarial inputs distributed outside of the regular inference space. As for the time step t in the objective, one step selected randomly out of each input will be adequate to achieve considerable performance. For the model with inputs always of hundreds of time steps, several more steps can be employed to increase the state inconsistency. Moreover, states of multiple layers $l \in L$ (L for all the layers) with respect to the same time step t will be leveraged to accelerate the search efficiency.

Coverage guidance. This part aims to cover the uncovered states, exercising more decision logics to produce adversarial inputs. RNN-Test leverages the proposed HS_C and CS_C metrics to guide adversarial testing, where CS_C guidance for LSTM models and HS_C for general RNN models. To boost the specific coverage, RNN-Test selects m hidden states or cell states to compose the optimization objective, as in formula (3). Rather than merely selecting uncovered states randomly, RNN-Test mainly chooses the states with values near to be covered so as to reach a higher coverage value at an earlier stage. Since CS_C is defined over a series of sections and the boundary sections are hardly covered, the states with values near the boundary section endpoints will be the targets to cover, thus leading RNN-Test to search in more sensitive space.

Subsequently, the joint optimization objective will be maximized by mutating the test inputs, unlike the training course minimizing the prediction error by tuning the parameters. Given the predefined objective, its derivative for the input x will be the perturbation, which is the gradient direc-

TABLE 1: Summary of RNN models to evaluate RNN-Test. The first four models are for the major evaluation. The last model MNIST LSTM is constructed for comparison to testRNN.

Model	Description	Architecture	Performance		
			Metric	Reported	Ours
PTB language model	General language model	Two-layer LSTM in its small configuration(i.e. fewer steps)	Train perplexity[1] Test perplexity	37.99 115.91	43.316 117.122
Spell checker model	Simple seq2seq model	Two-layer bi-direction LSTM for the encoder	Sequence loss	15%	10%
DeepSpeech ASR model	State-of-the-art ASR model	One-layer bi-direction LSTM with CNN layers around	WER[2]	16%	16%
MNIST GRU model	Handwritten digit recognition of GRU network	One-layer dynamic GRU	Test accuracy	-	96.5%
MNIST LSTM model	Handwritten digit recognition of LSTM network	One-layer LSTM	Test accuracy	98.3%	96.88%

[1] Perplexity, the universal metric for language models, where lower perplexity corresponds to a better model.
[2] Word error rate, a common performance metric for seq2seq and ASR models, where higher WER means worse predictions.

tion along which it increases or decreases most. Afterwards, the perturbations will be exploited to generate adversaries.

4.3 Adversarial input synthesis

For continuous inputs like speech, the perturbations could be applied directly to acquire the adversarial input. For NLP tasks whose inputs are words or characters scattered in discrete embedding space, procedure GEN_ADV is presented in Algorithm 1. In the procedure, we iteratively scale the gradient to be applied as the perturbation and then search for the nearest word/character in the embedding space to mutate the input step (lines 8 to 14 in Algorithm 1). This is a straightforward way to obtain valid adversarial inputs, ridding of embeddings which are not equivalent to legal words/characters. Besides, the embedding representations of words or characters in each NLP task are acquired after enough training, which could unveil their semantic properties. Therefore, searching along the gradient for the nearest embedding could get desired adversarial inputs with existing semantic information.

Algorithm 1 Adversarial input synthesis for discrete inputs

```
Input: $x \leftarrow$ original test input
       $t \leftarrow$ one time step selected to modify
       grad $\leftarrow$ perturbations obtained
       embs $\leftarrow$ embeddings of the vocabulary
       MAX_SCALE $\leftarrow$ maximum degree of scaling the gradient
1:  /*generate adversarial inputs for NLP tasks.*/
2:  procedure GEN_ADV($x$, $t$, grad, embs)
3:      $x' \leftarrow x$
4:      dist_vec $\leftarrow \emptyset$
5:      for scale $\in$ [1, MAX_SCALE] do  //search along the gradient
6:          pert = grad$_t$ $\times$ scale  //perturbation for the time step
7:          t_emb = $x_t$ + pert  //get invalid embedding by gradient ascent
8:          for emb $\in$ embs do
9:              dist = norm(t_emb - emb)  //distance of t_emb to emb
10:             dist_vec = dist_vec $\cup$ {dist}
11:         nearest_emb = argmin(dist_vec)  //the nearest embedding
12:         if nearest_emb != $x_t$ then
13:             $x'_t$ = nearest_emb  //modify the time step
14:             break
15:     return $x'$  //acquire the adversarial input
```

In the literature, the adversarial input is identified for the imperceptibility from the original input but with the distinct class label. In seq2seq domains with no classifications, it is hard to recognize a generated sequence as the adversarial input avoiding false positives, which has no standards yet [23], [24], [38]. Fortunately, model performance metrics are a good choice to exhibit qualities of adversarial inputs, which are supposed to be accessible in all the tasks. Consequently, adversarial inputs obtained will be fed into the model assessing whether to decay the performance and updating the coverage, where coverage information will be exploited to guide subsequent testing.

5 Experiment

5.1 Experiment Setup

Implementation. We developed the framework RNN-Test on the widely deployed framework Tensorflow 1.3.0, and evaluated RNN-Test on a computer having Ubuntu 16.04 as the host OS, with an Intel i7-7700HQ@3.6GHz processor of 8 cores, 16GB of memory and an NVIDIA GTX 1070 GPU.

As for hyperparameters in RNN-Test algorithms, such as m and MAX_SCALE, they are tuned for each tested model and not listed here for simplicity. We will release our code and datasets upon publication for further discussions.

Tested models. A summary of tested models is presented in Table 1. We mainly evaluated RNN-Test on the first four RNN models dealing with different tasks, three of which are seq2seq and one is seq2one. The last model MNIST LSTM is particularly constructed for comparison to testRNN. These models of common structures and various tasks provide more confidence for the generalization of RNN-Test to other RNN models.

PTB language model [27] is a well-known RNN model, basically to generate subsequent texts taking previous texts as input. It is the implementation of the fundamental LSTM [34] without particular adaptations for specific applications. The training and testing data are provided by the Penn Tree Bank dataset [39], and we extracted the first 25 sentences of the testing data for evaluation. We trained this model to achieve comparable performance to that reported using the same training course.

Spell checker model [28] is one of the widespread seq2seq models in NLP tasks, which receives a sentence with spelling mistakes as input and outputs the sentence with mistakes corrected. The training data are twenty popular books from project Gutenberg [40]. For testing, we constructed 160 sentences with spelling mistakes like examples of developers, thanks to rich sources from Tatoeba [41]. Since their pre-trained model is unavailable, we trained this model in the same way.

TABLE 2: Effectiveness of RNN-Test and other methods in their default settings, measured over adversarial inputs obtained by each method. Note that worse performance values (e.g. WER) indicate stronger test capability of methods (The best result across each row is denoted bold). The coverage guidance used by RNN-Test is given following w. (with), HS_C is hidden state coverage and CS_C is cell state coverage. The same symbols are used in below tables.

		Methodology					
Model	Performance	Original	Random testing	FGSM-based	DLFuzz-based	RNN-Test (w. HS_C)	RNN-Test (w. CS_C)
PTB language model	Perplexity	150.46	229.97	240.07	233.35	**285.13**	277.44
	Generation Rate[1]	-	**100.00%**	95.78%	93.59%	**100.00%**	**100.00%**
Spell checker model	WER	5.63	7.10	7.19	7.07	7.40	**7.49**
	BLEU[2]	0.870	0.830	0.829	0.826	0.827	**0.822**
	Success Rate[3]	-	64.58%	73.61%	73.61%	73.61%	**76.39%**
DeepSpeech ASR model	WER	5.50	5.35	6.65	6.18	**8.10**	7.80
	BLEU	0.796	0.800	0.747	0.786	**0.703**	0.720
	Success Rate	-	40.67%	90.00%	67.50%	**100.00%**	**100.00%**
MNIST GRU model	Accuracy	96.5%	42.22%	56.67%	46.67%	**20.00%**	-
	Success Rate	-	56.32%	41.37%	58.62%	**79.31%**	-

[1] Generation rate. Ratio of the test set the methodology has managed to produce the adversarial input.
[2] BLEU (Bilingual evaluation understudy). Correspondence of prediction and the ground truth, where higher BLEU means better predictions.
[3] Success Rate. Ratio of the generated adversarial inputs to successfully reduce the model performance, not used for the first model as its performance is recorded over all inputs.

DeepSpeech ASR model [29] is a state-of-the-art speech-to-text RNN model employed in lots of security-critical scenarios. Its pre-trained model DeepSpeech-0.1.1 (Mozilla's implementation) could be deployed conveniently, and our testing data are the first 20 samples extracted from the Common Voice corpus [42].

MNIST GRU model [30] is a seq2one classification model implemented with GRU network built for MNIST [43] dataset, a famous handwritten digit dataset. We followed [30] and constructed a one-layer GRU model with 96.5% accuracy, which is evaluated based on the first 30 images of MNIST test set.

Baselines. For comparison, we first customize adversarial testing methodologies FGSM [7] and DLFuzz [31] to work for RNN models. They both generate adversarial inputs by solving optimization problems in a gradient-based manner and achieve considerable efficiency. We implement their optimization objectives and coverage metrics on RNNs, while other procedures are the same as RNN-Test. For FGSM-based method, its optimization objective only contains the adversary search part without coverage guidance, whereas DLFuzz-based methodology also makes use of coverage guidance to obtain adversarial inputs, where neuron coverage (NC) is the underlying metric. Here, the NC definition of RNN models is the same as DeepTest [26]. Note that the customization will not degrade their performance since our optimized searching procedures are also used for them.

For relevant works on RNN testing, there is a significant gap to conduct comparisons due to framework incompatibility and open-source issues. Besides later release than our preliminary work, tested models of testRNN [22] and DeepStellar [21] are built on Keras and corresponding models on TensorFlow are mostly unavailable. Ultimately, we developed RNN-Test on an LSTM network [32] of MNIST dataset, which is an image classifier both evaluated in the two works. This MNIST LSTM network is constructed on TensorFlow and achieves comparable test accuracy over the default MNIST dataset. Furthermore, we also built DeepStellar to test and evaluate the PTB language model.

Research questions: We constructed experiments to answer the following research questions.

- **RQ1.** How is the effectiveness of the RNN-Test? (§ 5.2)
- **RQ2.** How is the effectiveness of coverage guidance for adversarial testing? (§ 5.3)
- **RQ3.** How is the quality of adversarial inputs obtained by RNN-Test? (§ 5.4)

5.2 Effectiveness of RNN-Test (RQ1)

To conduct a thorough evaluation, we compare our RNN-Test approach with other methodologies over the tested models, measuring the model performance fed with adversarial inputs obtained by each methodology. Besides, we also provide results of original test inputs, and those of random testing that randomly replaces a word/character of text input or applies Gaussian noise to speech and image input. We run them on each tested model over the same original test set three times, to alleviate the uncertainty each time. The same settings are adopted for below evaluations of other methods.

Overall results. Table 2 summarizes the overall results, from which we could derive the following inferences. Firstly, adversarial inputs can decline the model performance, since tested models all achieve worse performance over adversarial input sets than the original test sets.

Secondly, random testing methods can also obtain adversarial inputs, but they are far from satisfactory. For the 100% generation rate on PTB language model, random replacement could always get mutated inputs while FGSM- and DLFuzz-based methods may fail to find adversarial inputs for some inputs.

Thirdly, the RNN-Test approach outperforms FGSM-based and DLFuzz-based approaches, with more performance reduction and higher success (or generation) rate. For instance, in comparison with FGSM-based method, RNN-Test (w. HS_C) achieves 18.77% higher perplexity and 4.22% higher generation rate on PTB language model, 2.92% higher WER, 0.24% lower BLEU on spell checker model, 21.80% higher WER, 4.40% lower BLEU and 10% higher success rate on DeepSpeech ASR model, and 36.67% lower accuracy and 37.94% higher success rate on MNIST GRU model. As for the slight improvement on the spell checker model, the sparse embedding space may be the primary cause, since it largely limits the searching capability.

How to choose the appropriate coverage guidance. As shown in Table 2, RNN-Test guided with HS_C and CS_C both gain better effectiveness than other methodologies, with no one always superior to the other. When applied for realistic RNN tasks, it is straightforward to choose the appropriate coverage guidance. For LSTM models, both are good alternatives. For common RNN and GRU models, HS_C is the choice, since hidden states are universal across these structures while cell states are special of LSTM models.

Comparison to relevant RNN testing works. As described in § 5.1, we conduct comparisons to other RNN testing methods over an MNIST LSTM model due to framework incompatibility. Table 3 presents the comparison of RNN-Test to testRNN on MNIST LSTM model, both over 500 original inputs as the evaluation setting of testRNN. TestRNN employed random mutation (RM) and targeted mutation (TM) to generate 2000 test cases for evaluation, respectively. In RNN-Test, once one adversarial input is obtained for the corresponding test input, the testing procedure starts for another test input, and thus acquired 500 test cases. In Table 3, RNN-Test could generate much more adversarial inputs out of fewer test cases, obtaining 52.65% to 66.45% higher adversary rate than TM algorithm of testRNN, a refined means of RM. Although testRNN (RM) obtains smallest perturbations, testRNN (TM) with largest perturbations still reached limited adversary rate.

TABLE 3: Effectiveness of RNN-Test compared to testRNN on MNIST LSTM model over 500 original inputs. The results are listed in a similar way of testRNN [22], where those for testRNN are exactly that they reported.

MNIST LSTM model	**Methodology**			
	testRNN (RM)	testRNN (TM)	RNN-Test (w. HS_C)	RNN-Test (w. CS_C)
Test Cases Generated	2000	2000	500	500
# Adv. Inputs	26	63	**348**	279
Avg. Perturb. (L2 norm)	**1.051**	4.028	1.74	1.69
Adversary Rate	1.3%	3.15%	**69.6%**	55.8%

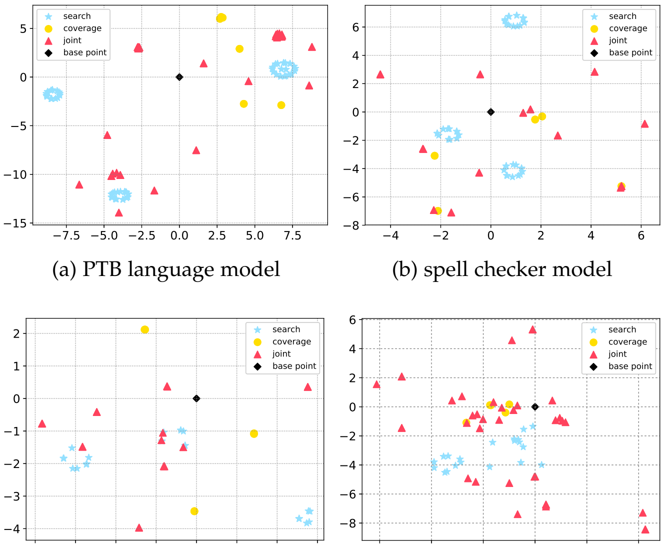
As to the state-of-the-art work DeepStellar, it reported that thousands of adversarial images were obtained for 100 test inputs after 6 hours on a high-performance server. Since DeepStellar adopts multiple random mutation strategies for each input seed with high iteration times, there are many adversarial inputs with huge distortion which are hard for humans to identify. Due to our design, it takes RNN-Test 16s to 24s to craft adversarial images for 100 inputs. Despite that, RNN-Test is also able to test each input continuously and generate much more adversarial inputs. If with a same high-performance server (with 28-core CPU, 196 GB RAM, and 4 NVIDIA Tesla V100 16G GPUs), RNN-Test is supposed to achieve comparable results.

Moreover, we adapted DeepStellar on PTB language model with their testing procedures and coverage criteria. In a similar way, we also mutate the text inputs by randomly replacing one word out of a sentence, with high mutation times for one input. As a result, DeepStellar generated more than 10000 adversarial inputs, obtaining 180.44 test perplexity with 84% generation rate. Due to numbers of input seeds, some inputs are hardly selected to mutate, with lower generation rate and guided coverage criteria (state-level criteria) of 50.43%. To conclude, RNN-Test acquires 53.76% to 58.02% more perplexity with 16% higher generation rate on PTB language model, compared with DeepStellar.

The answer to RQ1: The RNN-Test approach is effective in generating adversarial inputs, with the ability to reduce the model performance sharply with high success (or generation) rate.

5.3 State coverage guidance contributes to adversarial testing (RQ2)

Divergent perturbations of coverage guidance. The previous work [44] suggests that perturbations obtained by neuron coverage guidance are similar to adversary-based search methods (e.g. FGSM) and so the coverage guidance does not add too much, which is concluded based upon analyses over popular coverage guided testing methodologies [10], [12], [36]. But the conclusion may not work for the proposed state coverage metrics, since those criteria assessed are all over CNN neurons.



(a) PTB language model

(b) spell checker model

(c) DeepSpeech ASR model

(d) MNIST GRU model

Fig. 6: TSNE transformations of perturbations of the above approaches with different optimization objectives for one same test input. Search is adversary search, coverage is coverage guidance, joint is joint objective, same in below figures. The divergent distribution represents various perturbations and thus adversarial inputs.

Here, we recorded perturbation vectors obtained by approaches we evaluated over the same inputs of each RNN model. Besides their default settings, we run each approach with either pure adversary search or coverage guidance, as well as the joint way. To visualize, we leverage the state-of-the-art high-dimensional reduction technique TSNE [45] to

TABLE 4: Effectiveness of state coverage guidance, compared to neuron coverage guidance.

		Methodology		
Model	Performance	NC	HS_C	CS_C
PTB language model	Perplexity	150.46	**238.07**	236.96
	Generation Rate	0%	94.66%	**97.22%**
Spell checker model	WER	7.03	7.48	**8.00**
	BLEU	0.830	0.827	**0.823**
	Success Rate	73.61%	75.00%	**77.78%**
DeepSpeech ASR model	WER	5.45	**5.65**	5.35
	BLEU	0.801	**0.791**	0.794
	Success Rate	10.00%	40.00%	**55.00%**
MNIST GRU model	Accuracy	95.56%	**56.67%**	-
	Success Rate	1.15%	**44.83%**	-

transform multi-dimensional perturbation vectors into two dimensions. As Fig. 6 shows, there is no evident similarity of perturbations of the adversary search, coverage guidance or joint objectives. In contrast, the divergent distribution implies that coverage guidance is capable to offer alternative perturbations, whether utilized independently or jointly, thus providing varied adversarial inputs. Therefore, the coverage guidance is worthy to be applied to adversarial testing.

Effectiveness of pure coverage guidance for adversarial testing. State coverage guidance can also be adopted to discover adversarial inputs independently, due to the unique perturbations of coverage guidance. Table 4 presents the results of HS_C, CS_C, and NC as guidance to be the only optimization respectively. As shown, state coverage metrics as guidance can acquire adversarial inputs on the tested models, while NC guidance fails to obtain any on PTB language model. Overall, both CS_C and HS_C outperform NC as guidance, especially on PTB language model, DeepSpeech ASR model and MNIST GRU model. Therefore, neuron-based coverage metrics will not be appropriate for RNN models, as discussed in § 2.2. Surprisingly, when cross-referenced with Table 2 for the spell checker model, CS_C guidance exhibits the best performance with the highest WER and success rate.

Enhancement of coverage guidance to other methods. We also demonstrate that both FGSM-based and DLFuzz-based approaches with state coverage guidance could gain higher effectiveness than themselves and those jointed with NC guidance. For instance, Table 5 provides results of DLFuzz-based methodology jointed with HS_C and CS_C guidance, as well as its NC guidance. Compared with NC guidance, HS_C and CS_C guidance improve DLFuzz-based technique over the tested models in varying degrees. Additionally, similar results are attained for FGSM-based approach, where state coverage guidance improves more than NC guidance, as presented in Table 6. Thus, state coverage guidance is proven to be able to enhance other adversarial testing methodologies. Though these two methods could be improved with state coverage guidance, the most powerful means for most models is still the RNN-Test, as cross-referenced with Table 2.

In terms of MNIST GRU model, HS_C guidance could enhance the two methods to a large extent, even with higher effectiveness than RNN-Test. To be frank, RNN-Test with only state inconsistency optimization exhibits best performance over MNIST GRU model with 93.10% success rate and 6.67% accuracy. Despite that RNN-Test (w. HS_C or CS_C) is not always the best for all the models in this paper, our proposed methods including state inconsistency optimization as well as state coverage guidance are both worthwhile for further studies in RNN testing.

TABLE 5: Effectiveness of DLFuzz-based methodology with state coverage guidance, compared to its NC guidance.

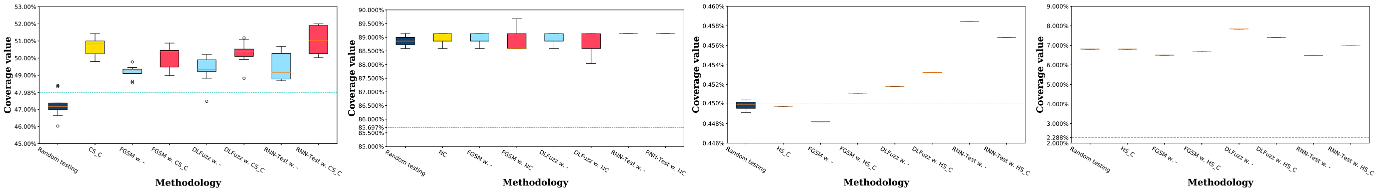
		Methodology (DLFuzz-based)		
Model	Performance	w. NC	w. HS_C	w. CS_C
PTB language model	Perplexity	233.35	**243.19**	238.71
	Generation Rate	95.42%	97.44%	**99.15%**
Spell checker model	WER	7.07	**7.44**	7.10
	BLEU	0.826	**0.825**	**0.825**
	Success Rate	73.61%	75.00%	**76.39%**
DeepSpeech ASR model	WER	**6.18**	6.15	6.10
	BLEU	0.786	0.785	**0.778**
	Success Rate	67.50%	**75.00%**	70.00%
MNIST GRU model	Accuracy	46.67%	**16.67%**	-
	Success Rate	58.62%	**82.76%**	-

TABLE 6: Effectiveness of FGSM-based methodology jointed with the coverage metrics, compared to its default setting with no coverage guidance.

		Methodology (FGSM-based)			
Model	Performance	w. -	w. NC	w. HS_C	w. CS_C
PTB language model	Perplexity	240.07	241.23	**256.91**	**256.91**
	Generation Rate	95.78%	98.69%	97.65%	**100.00%**
Spell checker model	WER	7.19	6.99	**7.46**	7.14
	BLEU	0.829	0.832	**0.828**	0.831
	Success Rate	**73.61%**	**73.61%**	**73.61%**	70.83%
DeepSpeech ASR model	WER	6.65	6.70	6.75	**6.80**
	BLEU	0.747	0.747	0.748	**0.746**
	Success Rate	**90.00%**	**90.00%**	**90.00%**	**90.00%**
MNIST GRU model	Accuracy	56.67%	53.33%	**20.00%**	-
	Success Rate	41.37%	44.83%	**79.31%**	-

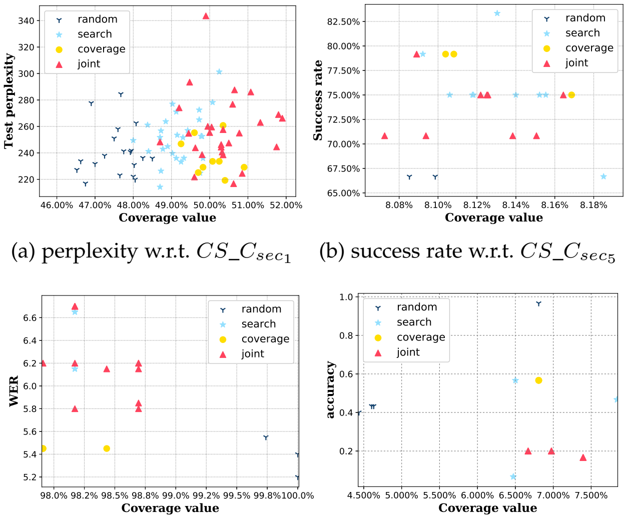
Coverage value may not be a strong indicator of methodology effectiveness. Numerous works [10], [12], [22], [36] adopt the coverage value as an indicator of effectiveness for adversarial testing. Meanwhile, researchers [44], [46], [47], [48] raised doubts that there may be limited correlations between coverage and robustness of DNNs.

Here, we have analyzed correlations between the model performance and values of coverage metrics on the first four models, but found out weak positive or negative correlations. As listed in Fig. 8, correlation values in four subfigures are 0.2314, -0.0667, -0.5362 and -0.3974, respectively (those not listed here are of analogous correlations. Therefore, we could not draw the conclusion that obtaining higher coverage definitely results in higher effectiveness in RNN testing. On the one hand, a higher coverage value does not ensure the method achieving a higher success rate or WER. On the other hand, adversarial input sets which highly decline the model performance (e.g. result in low perplexity) may not lead to higher coverages. This finding is consistent with similar studies in DNN testing [47], [48]. Though testRNN proved that their adversarial input set is with higher coverage rate than the normal input set, whether a higher coverage rate leads to a higher adversary rate also remains unsettled. Hence, we suggest that more efforts are demanded for adversarial testing with the coverage



(a) $CS\_C_{sec_5}$ on PTB model (b) NC on spell checker model (c) HS_C on DeepSpeech model (d) HS_C on MNIST GRU model

Fig. 7: Value ranges of coverage metrics among different approaches (w.- denotes no coverage guidance) over the same amount of adversarial inputs. The blue dashed lines denote the corresponding coverage value of original test input sets.



(a) perplexity w.r.t. $CS\_C_{sec_1}$ (b) success rate w.r.t. $CS\_C_{sec_5}$

(c) WER w.r.t. NC (d) accuracy w.r.t. HS_C

Fig. 8: The model performance metrics of adversarial input sets w.r.t. value of coverage metrics, where random is random testing. Subfigure (a), (b), (c) and (d) are for PTB language model, Spell checker model, DeepSpeech ASR model and MNIST GRU model, respectively.

guidance, but not just to improve the coverage value. Model performance indicators value more attention to evaluate an RNN testing means.

Simple illustration of value ranges of coverage metrics. Fig. 7 presents the value ranges of HS_C, CS_C and NC achieved by different methods on each tested model respectively. We also provide results of each methodology with and without the corresponding coverage guidance, since the coverage guidance still tends to improve the value. For each box, it represents a set of coverage values of the methodology at different times, marked with bounds and the median. Note that most boxes in Fig. 7c and Fig. 7d resemble lines because of the limited stochasticity and equal coverage values obtained on these models.

It must be claimed that coverage values strongly depend on the number of test inputs, and the same amount of inputs are supposed to be with similar value ranges. As presented, the value ranges of these coverage metrics among methodologies vary not much (within 5%), especially NC ranges are almost the same. It is the same case for figures not given here. Furthermore, HS_C values are always very low since it is inherently hard to boost HS_C with a few inputs, similar to boundary sections of CS_C over larger models. Meanwhile, it also supplies evidence that methodology effectiveness may be affected little by coverage values. However, coverage guidance is still worthy of more research investment. In summary, we could get the following answer.

> The answer to RQ2: State coverage metrics as guidance are able to acquire adversarial inputs, superior to neuron coverage guidance whether independently or jointed with adversary search. The coverage guidance has the potential to be more effective, since the divergent perturbations and best performance on the spell checker model.

5.4 Quality of adversarial inputs of RNN-Test (RQ3)

Samples of adversarial inputs. Table 7 lists samples of adversarial inputs on the two NLP models, with each approach to modify the same word. Due to only one word or character out of the sentence is modified for each test input to be the adversarial input, the generated adversarial sets will still remain natural and keep the semantics. Since the word to replace with is the nearest one along gradients across the embedding space, it is unlikely for abnormal words like "not" to be the target and substantially change the meaning.

For both models, RNN-Test tends to generate different words with other methods, offering diverse adversarial inputs. For PTB language model, our adversarial inputs could result in the model sampling words farther from semantics and generating higher perplexity texts, where prediction results of RNN-Test are with totally wrong semantics. Meanwhile, adversarial inputs for the spell checker model could result in the corrected mistakes in original inputs appearing again in the predictions of adversarial inputs.

For DeepSpeech ASR model, they could result in the model making wrong predictions, as depicted in Fig. 9. Such adversarial inputs of misleading semantics may harm the security requirements of tested models. Fig. 10 shows adversarial images on MNIST GRU model as an example, in which adversarial inputs fool the classifier while preserving the correctness for humans.

Imperceptible perturbations. Table 8 lists the averaged size of perturbations of each method on the tested models, namely the distortion introduced by perturbations applied. Note that l2 norm measures over the initial perturbations before applied according to various domain-specific constraints. For the first two models, we aim to modify the original input with one word or character replaced, which is

TABLE 7: Samples of adversarial inputs on the two NLP models, the targeted words to modify are in red and underlined. The affected results for the spell checker model are also underlined.

Methodology	PTB language model	spell checker model
Original	Input: no it was n't black Monday ... Perplexity: 259.67 Generate: economy goes forward on behalf of ...	Input: I would swim through theoocean just to see your smile again. Predict: I would swim through the ocean just to see your smile again. Input: The sound of yur voice islike siren's songto me. Predict: The sound of yur voice is like siren' song to me.
FGSM-based	Input: no it was n't black co. ... Perplexity: 376.46 Generate: economy goes forward on behalf of ...	Input: I would swim through theootean just to see your smile again. Predict: I would swim through the otean just to see your smile again. Input: The sound of yur voiceısliee siren's songto me. Predict: The sound of yur voice is liee siren' song to me.
DLFuzz-based	Input: no it was n't black due ... Perplexity: 357.38 Generate: soviets appear reluctant between france 's ...	Input: I would swim through theootean just to see your smile again. Predict: I would swim through the otean just to see your smile again. Input: The sound of yur voice islske siren's songto me. Predict: The sound of yur voice issle siren' song to me.
RNN-Test	Input: no it was n't black $ ... Perplexity: 513.91 Generate: soviets appear reluctant toward nov. a.m. ...	Input: I would swim through theoKcean just to see your smile again. Predict: I would swim through the cocean just to see your smile again. Input: The sound of yur voice isltke siren's songto me. Predict: The sound of yur voice istle siren' song to me.

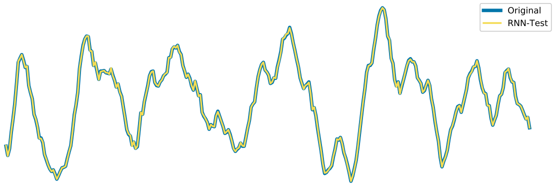


Fig. 9: An example adversarial input of RNN-Test for DeepSpeech ASR model. The waveform of a test input (blue, thick line) is overlapped with the waveform of the adversarial input (yellow, thin line). Each waveform is 500 samples long and was chosen randomly from the corresponding inputs. The original prediction is "the shop as closed on mondays" while the prediction for adversarial input is "the shop as close tan monas".

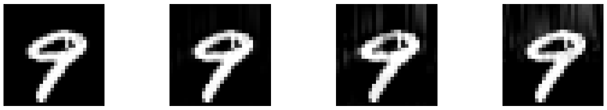

(a) Original (b) FGSM-based (c) DLFuzz-based (d) RNN-Test

Fig. 10: Samples of adversarial inputs for the same original one on MNIST GRU model. The original image is classified correctly as 9, whereas three adversarial images are classified as 4.

the nearest target found using the perturbations. Hence, all these methods will introduce the same distortion (one word or character) for the test input. Overall, FGSM-based approach achieves the smallest distortion over all the models, demonstrating its generality over DNN models. In addition, both DLFuzz-based and RNN-Test methods also induce minute distortions to achieve considerable effectiveness, ensuring the qualities of adversarial inputs.

During our evaluations, we found that our state inconsistency combined with state coverage guidance could generate large perturbations, implying the significance of states in RNN systems. For the first two models of discrete text inputs, scaling the gradient results in enormous perturbations. Since the modification is limited to one word or character, large perturbations will not lead to more distortion. For the last two models of continuous inputs, it is straightforward to utilize a relatively small learning rate before applying perturbations and obtain desired adversarial inputs.

TABLE 8: Distortion of adversarial inputs with respect to original inputs on four models (1 w for one word and 1 c for one character).

		Methodolgy				
Model	Distortion	Random testing	FGSM -based	DLFuzz -based	RNN-Test (w. HS_C)	RNN-Test (w. CS_C)
PTB language model	MED[1] l_2 norm[2]	**1 w** –	**1 w** **5.75**	**1 w** **5.75**	**1 w** 1322.92	**1 w** 1294.65
Spell checker model	MED l_2 norm	**1 c** –	**1 c** **0.0003**	**1 c** 0.26	**1 c** 25.68	**1 c** 55.17
DeepSpeech ASR model	l_2 norm	0.08	**0.0003**	0.03	0.04	0.02
MNIST GRU model	l_2 norm	0.34	**0.24**	0.30	0.41	-

[1] Minimum edit distance. Minimum number of single edits (substitutions here) required to change original input to the adversarial input.
[2] l2 norm. Relative size of initial perturbations to original inputs.

Concerning large perturbations on the two NLP models, we have attempted to restrict perturbations of all the methods by dividing with their absolute values, making all the perturbations less than 4.04. For PTB language model, after restriction, only RNN-Test w. CS_C still gets 1.7% higher perplexity than random testing except for all others. Furthermore, the results for spell checker model vary little after restriction. As a consequence, the RNN-Test approach is a good choice to craft adversarial inputs in high efficiency.

Time efficiency, and reality concerns. RNN-Test also has high time efficiency, producing each adversarial input costs 3s, 11s, 24s, and 1s on average on PTB language model, spell checker model, DeepSpeech ASR model, and MNIST GRU model, respectively. That means RNN-Test has the potential to be applied in industrial practice. As for realistic concerns about adversarial images of tasks like autonomous driving or image classification, adversarial cases will probably not encounter in real-world circumstances and weaken the security significance. In RNN testing, users are likely to mistype text inputs and fail NLP models in reality. Nevertheless, this is an urgent issue for ASR models at present, since adversarial audios [24] always become invalid when played over-the-air, which we will be devoted to in future works.

Improve the model by retraining. Last but not the least, adversarial inputs obtained by RNN-Test are also capable to improve the model performance by retraining. We tried on PTB language model and incorporated adversarial inputs (82.5 KB) to the training set (5.1 MB).

TABLE 9: The perplexity before and after retraining on PTB language model. Columns 3 and 5 are for the augmented training set. Columns 4 and 7 are for the improvement of retraining results w.r.t original results.

epoch	**train perplexity**			**valid perplexity**		
	original	w. adv.	increment	original	w. adv.	decrement
0	290.584	288.579	-0.690%	190.004	192.096	-1.101%
2	113.216	113.712	0.439%	140.328	140.339	-0.008%
4	86.290	87.195	1.049%	132.589	132.969	-0.287%
6	56.282	56.961	1.207%	121.410	120.566	0.695%
8	46.549	47.082	1.146%	122.981	121.611	1.114%
10	43.991	44.474	1.096%	123.065	121.385	1.365%
12	43.227	43.695	1.082%	122.440	121.020	1.159%

Table 9 presents the perplexity of PTB language model before and after retraining, where train perplexity indicates the performance on the training set while valid perplexity for the valid set. Here the data are averaged over 5 times of the same retraining process with 12 epochs, to mitigate affects due to the intrinsic indeterminism of neural networks. From columns 4 and 7, results show that the train perplexity of the model after retraining increases by 1.082% whereas the valid perplexity decreases by 1.159% in end. Moreover, the test perplexity after retraining is 102.75, which is also declined by 12.582% compared to the original test perplexity 117.53. Notice that even by incorporating fewer adversarial inputs (1.6KB), the valid perplexity still declines by 0.058%. Therefore, adversarial inputs could alleviate the over-fitting issue in training by reducing little train performance, but improving the valid and test performance and thus the robustness of RNN models.

The answer to RQ3: RNN-Test could efficiently produce adversarial inputs of high quality, declining the model performance sharply and improving the model by retraining.

6 Discussions

Target models and applications. RNN-Test is devoted to being general and scalable for variants of RNNs, but we could not exhaustively evaluate all the variants and applications. In this paper, we focus on seq2seq tasks based on common RNN systems such as LSTMs and GRUs. Next, our approach could work over not only tasks with continuous inputs like speech and images, but also those of discrete inputs like texts. Despite the special design for seq2seq tasks, RNN-Test could also apply to seq2one tasks like image classification.

When applied to other types of tasks in the future, formidable efforts are still required. Among the widespread domains of RNN systems, different tasks probably introduce particular challenges for testing. As to one-to-many tasks like music generation, it may require totally different ways to mutate the inputs rather than apply perturbations.

Taking another classification task (IMDB sentiment analysis GRU model [49]) as an example, RNN-Test reduced the model accuracy from 79.49% to 61.54% and achieved a success rate of 22.58%. Although RNN-Test can be applied to evaluate GRU models, specific task-specific constraints still limit its performance. We suggest that RNN-Test is a good choice for seq2seq tasks as well as image classification tasks.

Threats to validity. Though RNN-Test exhibits appreciable effectiveness with the default setting in evaluations, its performance is inevitably influenced by the parameters, including the number of states selected to boost, the weights applied to joint objectives and the scaling degree of perturbations, especially the ways of sections splitting of CS_C. They are worthy to be well explored in future work. Furthermore, the uncertainty running each time still exists, owing to stochastic word/character to modify, which could be diminished by fixing the target. Lastly, the structures of tested models are general to some extent, but training the spell checker model still costs hard work, due to its bad reproducibility of the training results given.

Limitations of RNN-Test. The RNN-Test approach mainly utilizes RNN states in the design and thus introduces consequent limitations. We summarize them as threefold. First, state wrapping aims to avoid interfering with model logics, but adaptation efforts may be necessary for some variants with complex structures. Second, hidden state coverage is defined for common RNN models mostly of fewer layers, which might not be applicable to stacked LSTMs. Similarly, cell state coverage is particularly designed for LSTM models. Third, due to large initial perturbations of our state inconsistency with state coverage guidance, it is usual though not heavy to finetune a learning rate for specific tasks.

7 Related Work

Adversarial deep learning. The concept of adversarial attacks was first introduced in [6]. It discovered that state-of-the-art DNNs would misclassify the input images by applying imperceptible perturbations, where these mutated inputs are called adversarial examples/inputs. Their work FGSM [7] and numerous following works [8], [9], [50] generate adversarial inputs by maximizing the prediction error in a gradient-based manner. They provide rich input resources for CNNs to improve their robustness.

Afterwards, [38] explains adversarial inputs for RNNs, but presents rough qualitative descriptions for those of sequential outputs. Recently, the security aspects of RNN-based tasks have drawn significant attention in the research communities, such as sentiment analysis [20], [51], [52], [53], text classification [52] and fake news detection [54]. These existing works mostly focus on tasks of categorical outputs. For those of text inputs, some works [19], [20] add, delete or substitute a word/character to construct adversarial inputs, leading models to give wrong classifications.

Unfortunately, few works are evaluating the high portion of RNN models processing sequential outputs. Due to no explicit class labels and no standards for such adversarial inputs, existing works attack these models to perform incorrectly in distinct ways. TensorFuzz [23] crafts adversarial inputs to lead the language model to sample words from the blacklist. Several works [24] fool well-known ASR models to produce targeted phrases given by the attacker. Another work [25] evaluates reading comprehension systems with inserted sentences to build adversaries altering semantics while answers unchanged. Other emerging methods perform specific attacks towards machine translation [55], question answering [56] or dependency parsing [57] tasks. Opposite to them, we aim to craft adversarial inputs that look nearly the same as original ones but highly decay the model performance. In this way, we propose RNN-Test as an effective and scalable methodology for diverse RNN systems, especially those of seq2seq tasks.

Coverage guided testing. Based upon the exposed threats of DNNs, traditional software testing techniques are subsequently applied to test DNN systems, where coverage guided testing is of a popular trend. DeepXplore [10] first introduces neuron coverage which is defined over CNN neurons with pre-defined thresholds. Then, DeepGauge [12] defines a set of coverage metrics with finer-grained granularity, where neuron value ranges are split as thousands of sections according to training data. DeepCT [36] is even fine-grained to measure over combinations of neuron outputs. As stated in § 2.2, these neuron-based coverage metrics can not be directly applied to RNN states.

As for works also among the first attempts of adversarial testing for RNN systems, DeepStellar [21] adapts coverage metrics of DeepGauge to test RNN models, which need to be abstracted as a Markov Chain first. Despite its effectiveness, it is inevitable to miss key features and introduce computation overhead owing to intrinsic properties of abstraction. Another work testRNN [22] designs novel coverage metrics according to structures of LSTM models, some of which are special to quantify temporal relations. Besides of similar classification tasks, these two works both mutate inputs directly (e.g. random noise) and use saturated coverage values to terminate testing. Our RNN-Test approach mutates inputs based on RNN logics and adopts coverage guidance to help search for adversarial inputs, which is effective for both seq2seq and seq2one domains.

8 Conclusions

We design and implement the adversarial testing framework RNN-Test for recurrent neural networks. RNN-Test focuses on testing the main RNN structures without limit to tasks, aggregating advantages of both the proposed search method and novel state coverage metrics as guidance. It is superior to existing methodologies for DNN testing and could effectively produce adversarial inputs over RNN models of various applications, reducing model performance evidently with high success (or generation) rate. We also first demonstrate that coverage guidance has a diverse searching capability for adversarial inputs compared with other methods and our state coverage guidance outperforms neuron coverage guidance in RNN testing.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, 2012.

[2] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 160–167. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390177

[3] G. Hinton, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, B. Kingsbury, and T. Sainath, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, November 2012. [Online]. Available: https://www.microsoft.com/en-us/research/publication/deep-neural-networks-for-acoustic-modeling-in-speech-recognition/

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, pp. 221–248, 2017.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Computer Science*, 2015.

[8] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[9] S. M. Moosavi Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, no. EPFL-CONF-218057, 2016.

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

[11] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

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

[13] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[14] P. Rodriguez, J. Wiles, and J. L. Elman, "A recurrent neural network that learns to count," *Connection Science*, vol. 11, no. 1, pp. 5–40, 1999.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations*, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[17] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 5528–5531.

[18] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[19] M. Sato, J. Suzuki, H. Shindo, and Y. Matsumoto, "Interpretable adversarial perturbation in input embedding space for text," *arXiv preprint arXiv:1805.02917*, 2018.

[20] S. Samanta and S. Mehta, "Towards crafting text adversarial samples," *arXiv preprint arXiv:1707.02812*, 2017.

[21] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "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*. ACM, 2019, pp. 477–487.

[22] W. Huang, Y. Sun, J. Sharp, W. Ruan, J. Meng, and X. Huang, "Coverage guided testing for recurrent neural networks," *arXiv preprint arXiv:1911.01952*, 2019.

[23] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," *arXiv preprint arXiv:1807.10875*, 2018.

[24] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," *arXiv preprint arXiv:1801.01944*, 2018.

[25] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2021–2031.

[26] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 303–314.

[27] (2016, Sep.) Variant of ptb word lm that could save and restore the model and display the predictions. https://github.com/nelken/tf.

[28] (2017, June) A seq2seq model that can correct spelling mistakes. https://github.com/Currie32/Spell-Checker/.

[29] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[30] (2018, May) Implementing lstm and gru networks using tensorflow (section 21). https://www.cnblogs.com/zyly/p/9029591.html.

[31] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 739–743.

[32] U. ZCC, "Understanding inputs and training process of lstm," *https://www.cnblogs.com/USTC-ZCC/p/11171209.html*, 2019.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[35] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[36] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang, "Combinatorial testing for deep learning systems," *arXiv preprint arXiv:1806.07723*, 2018.

[37] Y. Gong and C. Poellabauer, "Crafting adversarial examples for speech paralinguistics applications," *arXiv preprint arXiv:1711.03280*, 2017.

[38] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 2016, pp. 49–54.

[39] A. Taylor, M. Marcus, and B. Santorini, "The penn treebank: an overview," in *Treebanks*. Springer, 2003, pp. 5–22.

[40] (2016, Apr.) Free ebooks - project gutenberg. http://www.gutenberg.org/ebooks/search/?sort_order=downloads.

[41] (2006) Tatoeba is a collection of sentences and translations. https://tatoeba.org/cmn/downloads.

[42] (2017, June) Common voice is a project to help make voice recognition open to everyone. https://voice.mozilla.org/.

[43] Y. LeCun, "The mnist database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[44] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural coverage criteria for neural networks could be misleading," 2019.

[45] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[46] Y. Dong, P. Zhang, J. Wang, S. Liu, J. Sun, J. Hao, X. Wang, L. Wang, J. S. Dong, and D. Ting, "There is limited correlation between coverage and robustness for deep neural networks," *arXiv preprint arXiv:1911.05904*, 2019.

[47] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "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*, 2020, pp. 851–862.

[48] S. Yan, G. Tao, X. Liu, J. Zhai, S. Ma, L. Xu, and X. Zhang, "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*, 2020, pp. 775–787.

[49] (2017, Jul.) Bidirectional gru with attention mechanism on imdb sentimental analysis dataset. https://github.com/AllenCX/IMDB-RNN-Attention.

[50] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[51] T. Wang, X. Wang, Y. Qin, B. Packer, K. Li, J. Chen, A. Beutel, and E. Chi, "Cat-gen: Improving robustness in nlp models via controlled adversarial text generation," *arXiv preprint arXiv:2010.02338*, 2020.

[52] S. Garg and G. Ramakrishnan, "Bae: Bert-based adversarial examples for text classification," *arXiv preprint arXiv:2004.01970*, 2020.

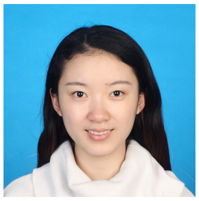
[53] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," *arXiv preprint arXiv:1812.05271*, 2018.

[54] T. Le, S. Wang, and D. Lee, "Malcom: Generating malicious comments to attack neural fake news detection models," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 282–291.

[55] Y. Cheng, L. Jiang, and W. Macherey, "Robust neural machine translation with doubly adversarial inputs," *arXiv preprint arXiv:1906.02443*, 2019.

[56] B. Wang, H. Pei, B. Pan, Q. Chen, S. Wang, and B. Li, "T3: Tree-autoencoder constrained adversarial text generation for targeted attack," *arXiv preprint arXiv:1912.10375*, 2019.

[57] X. Zheng, J. Zeng, Y. Zhou, C.-J. Hsieh, M. Cheng, and X.-J. Huang, "Evaluating and enhancing the robustness of neural network-based dependency parsing models with adversarial examples," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 6600–6610.

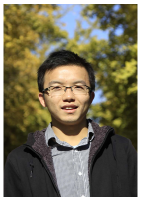


Jianmin Guo is a Ph.D. candidate at School of Software Engineering, Tsinghua University, Beijing, China. She received the BS degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. Her research interests are software testing, mainly focusing on deep learning testing and adversarial testing of recurrent neural networks.

Quan Zhang is a Ph.D. student at School of Software Engineering, Tsinghua University, Beijing, China. He received the BS degree in computer science from Beijing University of Posts and Telecommunications, Beijing, China, in 2020. His research interests are backdoor detection of deep learning systems.

Yue Zhao is a software engineer in Huawei Technologies Co., Ltd. He received MS degree at School of Software Engineering, Tsinghua University, Beijing, China, in 2020. He received the BS degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. His research interests are deep learning testing and backdoor detection of deep learning systems.

Heyuan Shi received the BS degree in school of information science and engineering, Central South University, Changsha, China, in 2015. He received the Ph.D degree in School of Software, Tsinghua University, Beijing, China, in 2020. His current research interests include software safety and security of embedded system, Internet of things and operating systems.



Yu Jiang received the BS degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2010, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2015. He was a Postdoc in the department of computer science of University of Illinois at Urbana-Champaign, IL, USA, in 2016. He is now an associate professor in Tsinghua University, Beijing, China. His current research interests include model driven design and program analysis.

Jiaguang Sun received the BS degree in automation science from Tsinghua University in 1970. He is currently a professor in Tsinghua University. He is dedicated in teaching and R&D activities in computer graphics, computer-aided design, formal verification of software, and system architecture. He is currently the director of the School of Information Science & Technology and the School of Software in Tsinghua University.