Coverage-Guided Testing for Recurrent Neural Networks

Wei Huang, Youcheng Sun, Xingyu Zhao, James Sharp, Wenjie Ruan, Jie Meng, and Xiaowei Huang

***Abstract*—Recurrent neural networks (RNNs) have been applied to a broad range of applications, including natural language processing, drug discovery, and video recognition. Their vulnerability to input perturbation is also known. Aligning with a view from software defect detection, this article aims to develop a coverage-guided testing approach to systematically exploit the internal behavior of RNNs, with the expectation that such testing can detect defects with high possibility. Technically, the long short-term memory network (LSTM), a major class of RNNs, is thoroughly studied. A family of three test metrics are designed to quantify not only the values but also the temporal relations (including both stepwise and bounded-length) exhibited when LSTM processing inputs. A genetic algorithm is applied to efficiently generate test cases. The test metrics and test case generation algorithm are implemented into a tool TESTRNN, which is then evaluated on a set of LSTM benchmarks. Experiments confirm that TESTRNN has advantages over the state-of-the-art tool DeepStellar and attack-based defect detection methods, owing to its working with finer temporal semantics and the consideration of the naturalness of input perturbation. Furthermore, TESTRNN enables meaningful information to be collected and exhibited for users to understand the testing results, which is an important step toward interpretable neural network testing.**

***Index Terms*—Coverage-guided testing, coverage metrics, recurrent neural networks (RNNs), test case generation.**

I. Introduction

FEEDFORWARD neural networks (FNNs), notably convolutional neural networks (CNNs), are vulnerable in various safety and security scenarios, subject to adversarial attack [43], backdoor attack [17], data poisoning attack [37], privacy issues [38], etc. These defects are extensible to recurrent neural networks (RNNs). In this article, we study the RNN defects, focusing on adversarial samples [1] and backdoor samples [10]. These defects will lead a well-trained RNN to mispredictions. Different from CNNs, RNNs exhibit particular challenges, due to their more complex internal structures and their processing of sequential inputs with a temporal semantics, supported by their internal memory components. A generic RNN layer takes a sequential sample x as input, updates its internal state c, and generates an output h. Other structural components may be required for specific RNNs. Given an input $\{x_t\}_{t=1}^{n}$, the RNN layer will be unfolded with respect to the size n of the input, and therefore each structural component has a corresponding sequence of representations, for example $\{h_t\}_{t=1}^{n}$. Such a sequence of representations form a temporal evolution.

Coverage-guided testing has achieved a great success in software defect detection, and has been extended to work with FNNs in, e.g., [25], [32], [40], [41], [49], where a collection of FNN coverage metrics can be found. The definitions of these metrics are based on the structural information, such as the neurons' activations [25], [32], the relation between neurons in neighboring layers [40], [41], etc. While existing coverage metrics for FNNs may be adapted to work with RNNs, they are insufficient because they do not work with the internal structures of RNNs and, more importantly, the most essential ingredient of RNNs—the temporal relation—is not considered. Moreover, we note that a few coverage metrics are proposed in [12] for RNNs, by making simple extensions to those of FNNs without considering the temporal relation and the internal structures (e.g., the important components of RNNs, such as gates). *This article is to develop dedicated coverage metrics for RNNs, to take into account the additional structures and temporal semantics.*

As suggested in [11] and [23], a test metric does not have to be strongly related to adversarial samples, a specific type of defects corresponding to the robustness requirement of a neural network. This is not surprising, and actually not new (for software testing). As stated in [5], a (software) program with high test coverage has more of its source code executed during testing, which suggests it has a lower chance of containing undetected software defects compared to a program with low test coverage. We concur with this view, and suggest that *instead of identifying a particular type of defects, such as adversarial samples, coverage-guided testing is to generate a set of test cases as diversified as possible while preserving the naturalness, in order to exploit the internal*

Manuscript received 21 March 2021; accepted 4 May 2021. Date of publication 10 June 2021; date of current version 1 September 2022. This work was supported in part by the U.K. EPSRC projects on Offshore Robotics for Certification of Assets under Grant EP/R026173/1 and End-to-End Conceptual Guarding of Neural Architectures (EnnCore) under Grant EP/T026995/1, in part by the U.K. Dstl projects on Test Coverage Metrics for Artificial Intelligence and Safety Argument for Learning-enabled Autonomous Underwater Vehicles, in part by the ORCA Partnership Resource Fund Toward the Accountable and Explainable Learning-Enabled Autonomous Robotic Systems, and in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant 956123. Associate Editor: J. Zhang. *(Corresponding author: Wei Huang.)*

Wei Huang, Xingyu Zhao, and Xiaowei Huang are with the University of Liverpool, Liverpool L69 3BX, U.K. (e-mail: W.Huang23@liverpool.ac.uk; Xingyu.Zhao@liverpool.ac.uk; Xiaowei.Huang@liverpool.ac.uk).

Youcheng Sun is with the Queen's University Belfast, Belfast BT7 1NN, U.K. (e-mail: Youcheng.Sun@qub.ac.uk).

James Sharp is with the Defence Science and Technology Laboratory, Salisbury SP4 0JQ, U.K. (e-mail: jsharp1@mail.dstl.gov.uk).

Wenjie Ruan is with the University of Exeter, Exeter EX4 4PY, U.K. (e-mail: w.ruan@exeter.ac.uk).

Jie Meng is with the Loughborough University, Loughborough LE11 3TU, U.K. (e-mail: J.Meng@lboro.ac.uk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2021.3080664.

Digital Object Identifier 10.1109/TR.2021.3080664

behavior of the neural networks that has real operational impact. The proposed coverage metrics in this article are of such desirable features of being *diverse and natural*—with increased coverage, our approach is more likely to find different types of faulty behaviors (e.g., adversarial samples and backdoor samples) that manifest at multiple small regions in the input space (rather than adversarial samples clustered in one region as what normally attack-based methods find). Especially when the operational profile is unknown or changing, such diversified test cases are of particular importance for improving the delivered reliability [4] (indeed, spending all the budget on testing one input region that potentially has limited chance to be operated in practice is unwise). Meanwhile, our diversified test cases are "closer" to their seeds (points on the RNN's data manifold), compared to other state-of-the-art tools, implying higher chance to be seen in the real-life operation, thus preserving the naturalness.

Contributions: We first discuss in Section III why the coverage-guided testing is useful in analyzing RNNs and how to reasonably define the effectiveness of a testing framework. We focus on long short-term memory networks (LSTMs), which is the most important class of RNNs, and design three LSTM structural coverage metrics, namely boundary coverage (BC), stepwise coverage (SC), and temporal coverage (TC). Simply speaking, TC quantifies the multistep temporal relation, which describes the internal behavior on how LSTM cell processing inputs, whereas BC and SC quantify the value and single-step change of the temporal relation, respectively. We also discussed in Section V how to position the new metrics against a few closely related techniques, such as complete verification techniques, existing metrics, etc.

We implement the proposed coverage metrics into a prototype tool TESTRNN,[1] which includes two algorithms—a random mutation and a genetic algorithm based targeted mutation—for test case generation. In particular, targeted mutation uses the coverage knowledge to guide the test case generation. Initially, a random mutation is taken to generate test cases. Once the untargeted randomization has been hard to improve the coverage rate, a targeted mutation by considering the distance to the satisfaction of unfulfilled test conditions is taken to generate corner test cases.

We conduct an extensive set of experiments over a wide range of LSTM benchmarks to confirm the utility of TESTRNN and the proposed coverage-guided RNN testing approach from the following aspects.

1) Diversity of generated test cases (see Section VII-B), with the observations that the LSTM model's functional coverage can be approximated using our structural coverage metrics (see Section VII-B1) and our metrics complement existing metrics in guiding the exploitation of the input space (see Section VII-B2).
2) Detecting defects (see Section VII-C), with the observations that TESTRNN can not only find adversarial behaviors for the robustness of RNNs (see Section VII-C1) but also identify backdoor inputs for the security of RNNs (see Section VII-C2).
3) Usefulness of test case generation (see Section VII-D), with the observation that TESTRNN is efficient and effective in achieving high coverage rates (see Section VII-D).
4) Comparison with dedicated defect detection (see Section VII-E), with the observation that our test method can find a set of more diversified adversarial samples, and these samples are more likely to occur in real world.
5) Comparison with state-of-the-art tool DeepStellar (see Section VII-F), with the observations that our metrics are better at guiding the exploitation of the input space and TESTRNN may achieve good coverage on the metrics in DeepStellar but not vice versa.
6) Exhibition of LSTM internal working mechanism (see Section VII-G), with the conclusion that semantic meanings behind the test metrics can help users understand the learning mechanism of LSTM model, making a step toward interpretable LSTM testing.

The organization of this article is as follows. Section II gives the preliminaries. We will discuss the rationale of coverage-guided testing in Section III. After this, we present our proposed test metrics in Section IV. This is followed by discussing in Section V how these new metrics are related to the complete verification techniques, existing coverage metrics, and adversarial defense techniques. We present our test case generation algorithm in Section VI and the experimental evaluation in Section VII. We review related works in Section VIII. Finally, Section IX concludes this article.

II. RNN Preliminaries

FNNs model a function $\phi : X \to Y$ that maps from input domain X to output domain Y: given an input $x \in X$, it outputs the prediction $y \in Y$. For a sequence of inputs $x_1, \ldots, x_n$, an FNN ϕ considers each input individually, that is, $\phi(x_i)$ is independent from $\phi(x_{i+1})$.

By contrast, an RNN processes an input sequence by iteratively taking inputs one by one. A recurrent layer can be modeled as a function $\psi : X' \times C \times Y' \to C \times Y'$ such that $\psi(x_t, c_{t-1}, h_{t-1}) = (c_t, h_t)$ for $t = 1 \ldots n$, where t denotes the t^{th} time step, c_t is the cell state used to represent the intermediate memory, and h_t is the output of the t^{th} time step. More specifically, the recurrent layer takes three inputs: x_t at the current time step, the prior memory state c_{t-1}, and the prior cell output h_{t-1}; consequently, it updates the current cell state c_t and outputs hidden state h_t.

RNNs differ from each other given their respective definitions, i.e., internal structures, of recurrent layer function ψ, of which LSTM in (1) is the most popular and commonly used one

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ c_t &= f_t * c_{t-1} + i_t * \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(c_t). \end{aligned} \tag{1}$$

[1] [Online]. Available: https://github.com/TrustAI/testRNN

In LSTM, σ is the sigmoid function and $\tanh$ is the hyperbolic tangent function; W and b represent the weight matrix and bias vector, respectively; f_t, i_t, and o_t are internal gate variables of the cell. In general, the recurrent layer (or LSTM layer) is connected to nonrecurrent layers, such as fully connected layers, so that the cell output propagates further. We denote the remaining layers with a function $\phi_2 : Y' \to Y$. Meanwhile, there can be feedforward layers connecting to the RNN layer, and we let it be another function $\phi_1 : X \to X'$. As a result, the RNN model that accepts a sequence of inputs $x_1, \ldots, x_n$ can be modeled as a function φ such that $\varphi(x_1 \ldots x_n) = \phi_2 \cdot \psi(\prod_{i=1}^{n} \phi_1(x_i))$.

Cell structure: The processing of a sequential input $x = \{x_t\}_{t=1}^n$ with an LSTM layer function ψ, i.e., $\psi(x)$, can be characterized by gate activations $f = \{f_t\}_{t=0}^n$, $i = \{i_t\}_{t=0}^n$, $o = \{o_t\}_{t=0}^n$, cell states $c = \{c_t\}_{t=0}^n$, and outputs $h = \{h_t\}_{t=0}^n$. We let $\mathcal{S} = \{f, i, o, c, h\}$ be a set of structural components of LSTM, and use variable s to range over $\mathcal{S}$.

Sequential structure: Each s represents one aspect of the concrete status of an LSTM cell. To capture the interactions between multiple LSTM steps, temporal semantics are often used to understand how LSTM performs [29]. Test metrics in this article will rely on the structural information, such as *aggregate knowledge* ξ_t^h and *remember rate* $\xi_t^{f,avg}$, as explained below, and their temporal relations.

Output h is seen as short-term memory (as opposed to c for long-term memory) of LSTM. It is often used to understand how information is updated, either positive or negative according to the value of h_t. Thus, we have

$$\begin{aligned} \xi_t^{h,+} &= \sum\{h_t(j) | j \in \{1, \ldots, |h_t|\}, h_t(j) > 0\} \\ \xi_t^{h,-} &= \sum\{h_t(j) | j \in \{1, \ldots, |h_t|\}, h_t(j) < 0\} \\ \xi_t^{h} &= |\xi_t^{h,+} + \xi_t^{h,-}|. \end{aligned} \tag{2}$$

Intuitively, ξ_t^h represents the *aggregate* knowledge regarding short-term memory.

The forget gate f is a key factor for long-term memory in LSTM, as it controls whether the aggregate information can be passed on to the next (unfolded) cell or not. The portion of information passed is then measured by $\xi_t^{f,\text{avg}}$ as follows:

$$\xi_t^{f,\text{avg}} = \frac{1}{|f_t|} \sum_{j=1}^{|f_t|} f_t(j) \tag{3}$$

Example 2.1: Fig. 1 presents a set of visualizations to the temporal update of the abstract information. In particular, the top row contains curves for $\mathcal{N}_z(\xi_t^h)$ and the second row contains curves for $\mathcal{N}_m(\xi_t^{f,\text{avg}})$, changed with respect to the time. The third row visualizes the evolution of stepwise change information $\mathcal{N}_m(\Delta\xi_t^h)$. $\mathcal{N}_z$ and $\mathcal{N}_m$ are two normalization function, which will be introduced later.

Let $\mathcal{A} = \{+, -, \text{avg}\}$ be a set of symbols representing the abstraction functions as in (2) and (3). The above can be generalized to work with any $s \in \mathcal{S}$ and $a \in \mathcal{A}$. For $\xi_t^{s,a}$, once given a fixed input x, we may write $\xi_{t,x}^{s,a}$.

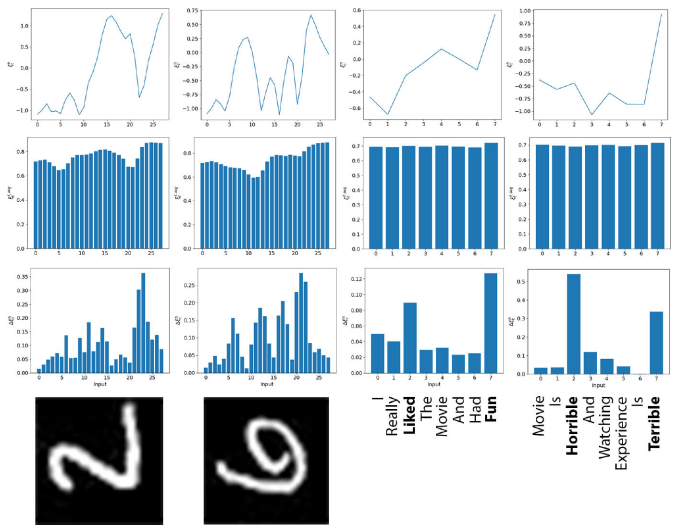

Fig. 1. Examples to show how positive and negative elements of output vectors represent the information in MNIST and IMDB models. The x-axis includes the inputs (bottom row) and the y-axis includes $\mathcal{N}_z(\xi_t^h)$ (top row), $\mathcal{N}_m(\xi_t^{f,\text{avg}})$ (second row), and $\mathcal{N}_m(\Delta\xi_t^h)$ (third row) values. In MNIST, each column of pixels corresponds to a step in LSTM and in the IMDB model, each step represents a word in the movie review.

III. Problem Statement

This section explains why the coverage-guided testing is useful in analyzing RNNs and how to reasonably justify its effectiveness. Fig. 2 (Left) presents the connection between verification and testing in this context. While incomplete, testing has been shown practical—and in many cases, sufficiently effective—in providing assurance to the quality of software. As in Fig. 2, these two approaches overlap on the "Flaws." Verification can detect defects because it exhaustively exploits all internal behavior of RNNs, which include defective behaviors. On the other hand, testing approach uses test cases to approximate—or sample—the internal behavior. Due to the finite sampling, defects may or may not be detected. Therefore, testing needs to be systematic to be effective in defect detection, and coverage-guided testing is one of the main approaches.

Due to the size and the temporal semantics of RNNs, it becomes important to find a (meaningful) set of metrics to guide the sampling or the test case generation. Our proposed set of coverage metrics plays the role of such guidance—as suggested in Fig. 2 (Left), coverage-guided testing generates a set of test cases to exploit the internal behavior of the neural networks. We note that such coverage metric does not have to be strongly correlated to adversarial samples [11], [23] (a specific type of defects corresponding to the robustness requirement of a neural network). What really matters is, within the testing budget, to find defects that are as diverse and natural as possible so that fixing them would gain maximized impact on the delivered reliability.

There are two main goals of testing [13]: debug testing, which probes the software for defects, and operational testing, which is to gain confidence that the software is reliable [55]. The former seeks test cases to excite as many failures as possible

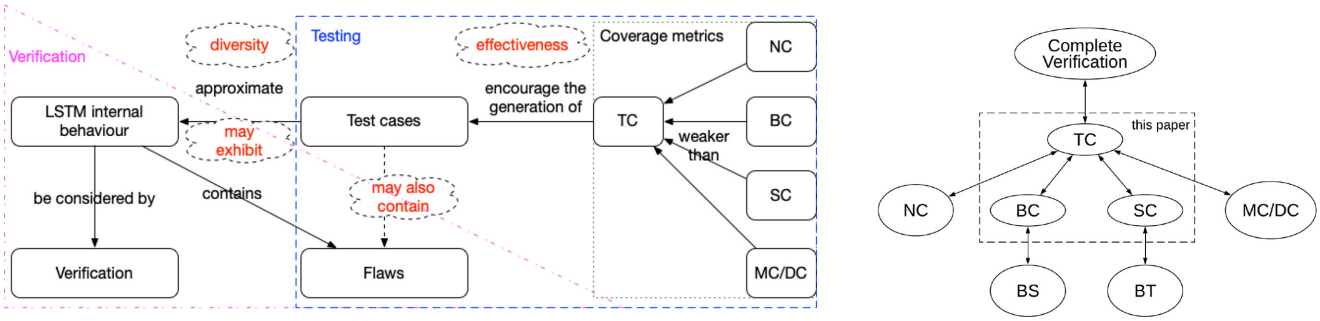

Fig. 2. (Left) Connection of verification and coverage-guided testing frameworks. Verification and testing overlap on "Flaws," representing that they have the same objective. From verification to testing, an approximation is made, i.e., test cases approximate the LSTM internal behavior. Guidelines (colored with red) are needed to ensure the approximation quality. (Right) Relation between coverage metrics. NC: neuron coverage [32], BS: basic state coverage [12], BT: basic transition coverage [12], and MC/DC: modified condition/decision coverage [40]. Arrows represent the "weaker than" relation between metrics.

(then we may fix the defects behind them), but the test cases normally are not representative of the software's day-to-day operation. Confidence in the delivered reliability can only be gained by the later, i.e., testing that represents the typical usage (the operational profile) [20]. Coverage-guided testing belongs to the former, whereas our method should also be designed in the best interest for the operational reliability. That is, our test cases should be more likely generated from the high probability density area on the operational profile, compared to attack-based and other state-of-the-art debug testing methods, so that the defects found are more "practical" in the sense that fixing them would effectively improve the operational reliability.

The question is on how effective a specific testing approach is when exploiting the internal behavior. Below, we provide a few guidelines by evaluating the connections of entities in Fig. 2 (Left) (shown with red color).

First, the test cases are required to be *diversified and natural*, so as to cover the LSTM internal behavior comprehensively. This is to avoid the test cases being clotted together in a small region of the input space (representing a certain type of defects) and lose the ability of finding other defects manifested in different regions. However, it is easy to generate diversified test cases by "forcing diversity" (e.g., by selecting inputs that maximize the average interpoint distance). Thus, diversity criteria is only sensible when paired with naturalness—test cases should not be far from the RNN's data manifold (i.e., potentially high density area of its future optional profile). Second, the test cases can reveal defects. While it is hard to establish strong correlation between test metrics and a specific type of defects, it is still a reasonable request that the generated test cases *reveal* defects as many as possible. Third, the test case generation algorithm needs to be effective, in terms of its ability in improving the coverage rate with diversified and natural test cases. Finally, to show that the generated test cases are sufficiently representative, we may use the test cases to exhibit the working mechanism of LSTM.

Moreover, given the complexity of the internal behavior in RNNs, we believe a set of coverage metrics are needed to ensure that the aforementioned guidelines can be achieved. In an ideal case where the testing budget is sufficient, our metrics in this article may complement—instead of replace—others, and vice versa. These guidelines will be used when designing our experiments in Section VII.

IV. LSTM Test Coverage Metrics

In this section, we present a family of three coverage metrics (BC, SC, and TC) for the testing of LSTM models. These metrics take into account both the values of structural information $\xi_t^{s,a}$ for $s \in \mathcal{S}$ and $a \in \mathcal{A}$ as in (2) and (3) and their stepwise and bounded-length temporal relations. We utilize two normalization methods for the convenience of determining thresholds, independent of specific dataset. $\mathcal{N}_z$ and $\mathcal{N}_m$ are z-score and min–max normalization function defined as follows:

$$\mathcal{N}_z(\xi) = \frac{\xi - \mu}{\sigma}, \quad \mathcal{N}_m(\xi) = \frac{\xi - \min}{\max - \min}$$

where parameters μ, σ, min, and max can be derived from the training dataset. The z-score normalization is suitable for preprocessing test conditions quantifying the relations between different features, e.g., TC, while min–max normalization is better for the test conditions to hit the large activation values, e.g., BC and SC. Given a time series of length n, we can choose the sequence of interest $[t_1, t_2]$ ($t_1 \geq 1$ and $t_2 \leq n$) to implement the following coverage metrics.

A. Boundary Coverage

Boundary values are often regarded as important cases in software testing, as they could exploit extreme software behaviors. We therefore define BC for depicting test conditions that cover the boundary values of the LSTM data flow as follows:

$$\{\mathcal{N}_m(\xi_t^{s,a}) \geq \alpha_{\max},\ \mathcal{N}_m(\xi_t^{s,a}) \leq \alpha_{\min} | t \in \{t_1 \cdots t_2\}\}.$$

Thresholds $\alpha_{\max}$ and $\alpha_{\min}$ are chosen from interval $[0, 1]$. The min and max values can be estimated using values computed over the training dataset $\{\xi_{t,x}^{s,a} | t \in \{t_1 \cdots t_2\}, x \in \mathcal{D}_{\text{train}}\}$.

Example 4.1: Suppose that there is a test condition $\mathcal{N}_m(\xi_t^{i,\text{avg}}) > 0.9$. It requires that the $\mathcal{N}_m(\xi_t^{i,\text{avg}})$ value is greater than threshold 0.9. Intuitively, this condition exercises LSTM's learning ability on the input at time t. As (1) shows, the input gate i controls how much information from the input is received by the network: $\mathcal{N}_m(\xi_t^{i,\text{avg}}) = 1$ implies that all its information

is added to the long-term memory c and $\mathcal{N}_m(\xi_t^{i,\text{avg}}) = 0$ implies that no input information is added.

B. Stepwise Coverage

SC characterizes the temporal changes between connected cells. We use $\Delta\xi_t^s = |\xi_t^{s,+} - \xi_{t-1}^{s,+}| + |\xi_t^{s,-} - \xi_{t-1}^{s,-}|$ to outline the maximum change of the structural component $s \in \mathcal{S}$ at time t. E.g., $\Delta\xi_t^h$ is the change of short-term memory at time t. Then, the SC test conditions are defined as follows:

$$\{\mathcal{N}_m(\Delta\xi_t^s) \geq \alpha_{\text{SC}} | t \in \{t_1 \cdots t_2\}\}.$$

This set defines test conditions for LSTM's stepwise updates that exceed a threshold α_{SC}. Parameters for $\mathcal{N}_m$ are derived from $\{\Delta\xi_{t,x}^s | t \in \{t_1 \cdots t_2\}, x \in \mathcal{D}_{\text{train}}\}$.

Example 4.2: The intuition behind SC is to capture these significant inputs to the LSTM. As shown by the sentiment analysis LSTM example in Fig. 1 (Right), given two inputs, sensitive words "like", "horrible", "fun" trigger greater $\mathcal{N}_m(\Delta\xi_t^h)$ values than words "movie", "really", and "had."

C. Temporal Coverage

While the power of LSTM comes from its ability to memorize values over arbitrary time intervals, its test metrics need to ensure that the temporal patterns of memory updates are fully tested. This is essentially a time series classification problem and is intractable. In this part, we define test conditions to exploit temporal patterns of bounded length. Different from dynamic systems where the temporal relation can be infinite [8], the temporal relations in RNNs are always finite, because of the finite-sized input. Therefore, the bounded length does not lower the expressiveness of the test conditions. In particular, to facilitate the enumeration of all test conditions, we refer to symbolic aggregate approximation (SAX) [24] to convert any complicated time series of length v (v is usually a large number) into a symbolic sequence of length w ($v >> w$).

First of all, given any temporal curve $\xi^s = \{\xi_t^s\}_{t=t_1}^{t_2}$, we can reduce the dimension of temporal sequence from $v = t_2 - t_1$ to w following piecewise aggregate approximation (PAA):

$$\hat{\xi}_j^s = \frac{w}{v} \sum_{t=\frac{v}{w}(j-1)+t_1}^{\frac{v}{w}j+t_1} \xi_t^s. \tag{4}$$

The main idea of PAA is to approximate the original time series by splitting them into w equal sized segments and average the values in each segment. For example, the temporal curve in Fig. 3 is split into $w = 5$ dimensions. The new approximated curve is denoted as $\hat{\xi}^s = \{\hat{\xi}_j^s\}_{j=1}^w$.

Then, we can define the symbolic representation for the temporal curve after dimensionality reduction. We start from z-normalizing $\hat{\xi}^s$ and discretizing $D(\mathcal{N}_z(\hat{\xi}^s))$—the domain of $\mathcal{N}_z(\hat{\xi}^s)$—into a set Γ of subranges. This discretization can refer to the distribution of $\mathcal{N}_z(\hat{\xi}_j^s)$, which can be estimated by conducting probability distribution fitting over the training dataset (since $\hat{\xi}^s$ is z-normalized, $\mathcal{N}_z(\hat{\xi}_j^s)$ is subject to the standard normal distribution). Then, every normalized time series $\{\mathcal{N}_z(\hat{\xi}_j^s)\}_{j=1}^w$ can be represented as a sequence of symbols in the standard way. For example, in Fig. 3, the continuous space of $\mathcal{N}_z(\hat{\xi}^{s,a})$ is split into a set of three subranges $\Gamma = \{a, b, c\}$.

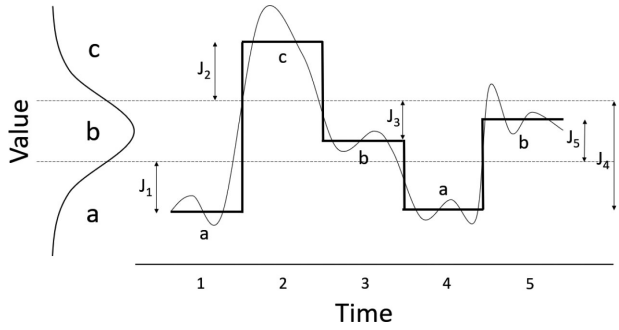

Fig. 3. Illustration of projecting a temporal curve (Gaussian distribution) into a sequence of symbols $acbab$.

Finally, test conditions from TC for covering a set of symbolic representations across multiple time steps $[t_1, t_2] \subseteq [1, n]$ can be expressed as follows:

$$\{\ell_1 \ell_2 \ldots \ell_w | \ell_j \in \Gamma, j \in [1, w]\}. \tag{5}$$

Essentially, TC requests the testing to meet a set of temporal patterns for a specific time span $[t_1, t_2]$. The total number of temporal patterns for TC to cover is $|\Gamma|^w$. We remark that with the help of SAX, test conditions in TC is scalable in tackling the complexity of time series.

Example 4.3: Fig. 1 (top row) demonstrates the temporal curve of hidden memory for each input across a selected time span. The curve is for $\mathcal{N}_z(\xi_t^h)$ and it is a clear illustration on the information processing of LSTM for each input. Fig. 3 further shows how a time series is converted into its symbolic representation $acbab$ with $\Gamma = \{a, b, c\}$ and $w = 5$.

V. Relation With RNN Defects

The aforementioned three coverage metrics encourage the exploration of the LSTM internal behavior, which is helpful in detecting the RNN defects. In this section, we discuss the rationale behind by referring to the general relation [see Fig. 2 (Right)] between our new coverage metrics, the complete verification, and a few existing coverage metrics. All discussions are supported by our experiments in Section VII.

A. Comparing With Complete Verification Techniques

For an input of length n, we denote the collection of curves (as in Fig. 3) for f, i, o as $Curv$. It precisely identifies an output (extracted features of the LSTM layer) and corresponds with a set of inputs (which cannot be differentiated by the LSTM layer). Let C_f, C_o, and C_i be the (possibly infinite) set of possible curves for their respective gates f, o, and i. We have Curv $= C_f \times C_o \times C_i$. We also have curves C_h and C_o, which can be obtained from Curv according to (1).

A complete verification method determines if there is an input that can lead to any unexpected behavior. That is, it is equivalent to determine if there is a combined curve in Curv that leads

to the unexpected classification.[2] To this end, TC (and its test case generation) can be seen as an approach to exhaustively, but discretely, explore one of the curve sets C_s for $s \in \mathcal{S}$. Therefore, while the combination of TC working on different gates may provide a complete verification, in general, the exploration of internal behavior through TC is a necessary, but insufficient, approach for verification.

B. Comparing With Other Coverage Metrics

The purpose of TC is to encourage the exploration of either C_f, C_o, or C_i with the generated test cases. However, such exploration may be computational intensive—the complexity is exponential with respect to the length n. Since BC and SC do not consider the temporal relation between steps or boundary values, they are computationally more manageable. Assuming that under ideal parameter settings (e.g., thresholds α_{max} and α_{SC} and the set Γ of symbols), we say that a metric A is weaker than another B if for any test suite, it cannot have a lower coverage rate w.r.t. A than that of B. It is not hard to see that both BC and SC are weaker than TC, and BC and SC are incomparable, as shown in Fig. 2 (Right).

Besides the new BC and SC, TC is stronger than existing coverage metrics that are originally proposed for CNNs. For example, NC [32], which requires the coverage of neurons whose value is over a threshold (e.g., 0 for ReLU activation function), can be adapted to work with say the gate value or hidden state value. In this case, it is weaker than TC and incomparable with SC. Although NC and BC have similar formal expressions, BC concerns the boundary value rather than a value that indicates the activation status. For the MC/DC metrics [40], they can be adapted to work between time steps in the new context of RNNs. With such adaptation, MC/DC encourages the exploration of relations between time steps, and therefore are weaker than TC.

DeepStellar [12] abstracts the evolution of hidden states of an RNN into a discrete-time Markov chain (DTMC) before considering state and transition coverage. Its basic state coverage (BS) and basic transition coverage (BT) are designed to cover the possible state values and possible transitions. Given that the DTMC is an abstraction of the curves C_h, BS and BT are both weaker than BC and SC, respectively.

Remarkably, the relations in Fig. 2 (Right) are based on theoretical analysis under "ideal parameter settings." They do not hold for any parameter settings. In our experiments in Section VII, we might observe the coverage rates on the same test set and the aforementioned relations are not in alignment, because of the specific parameter settings used (cf., Table I).

C. Defense Techniques for Robustness and Security

Some effective adversarial defense techniques, e.g. [9], are based on the observation that the adversarial samples exhibit different internal behavior to those behavior of training data samples. For security concerns, such as backdoor attack, activation patterns are also considered in the detection techniques, such as [6]. Consequently, with the exploration of more RNN internal behavior other than those appeared in the training data, it is more likely that RNN defects will be exposed.

[2]As shown in the experiments, an unexpected classification can be normal misclassification or caused by, e.g., backdoor attacks and adversarial inputs.

VI. Coverage-Guided Test Case Generation

Coverage metrics in Section IV define test conditions that request particular patterns of long/short-term updates of abstracted information across multiple LSTM time steps. Given an LSTM network and a specific test metric, the *coverage rate* denotes the percentage of test conditions that have been satisfied over a set of test cases, i.e., test suite. To more efficiently achieve high coverage rate, in this section, we develop the coverage-guided test case generation, as outlined in Fig. 4. We remark that, although focus of this article is LSTM, the proposed testing approach (including both test metrics and tests generation) can be extended to work with other kinds of RNNs, which use customized recurrent layer structures.

The TestRNN test case generation algorithm is detailed in Algorithm 1. The test suite $\mathcal{T}$ is initialized with $\mathcal{T}_0$, a corpus of seed inputs (Line 1). New test cases are generated by mutating seed inputs. It keeps the traceability of test cases via a mapping $orig$ that maps each test case generated back to its seed origin.

The main body of Algorithm 1 is a loop (Lines 5−10) that iterates unless some target coverage level is reached (Line 4). At each iteration, a test input x is selected from the input corpus $\mathcal{T}$ (Line 5), and it is mutated following the predefined mutation function m (Line 6). Newly generated test inputs are added into $\mathcal{T}$ (Line 7), where they are queued for the next iteration. If the generated test case does not pass the oracle (see Section VI-C1), it represents a defect and it is added to $\mathcal{T}_{\text{adv}}$ (Lines 9 and 10).

A. Selection Policies and Queuing

Not all inputs in the corpus $\mathcal{T}$ are equivalently important, and they are ranked once added to the input queue (as illustrated in Fig. 4). When sorting queuing inputs on $\mathcal{T}$ for the mutator engine, TestRNN particularly prioritizes two kinds of test inputs: those that are promising in leading to the satisfaction of unfulfilled test conditions and those that can trigger erroneous behaviors.

Thanks to its modular design, new selection policies can be easily integrated into TestRNN as plug-ins (as indicated by cloud shapes in Fig. 4). The design of TestRNN also features its high parallelism. The use of dynamically allocated input queues further optimizes its runtime performance.

B. Mutation Policies

The mutator engine lays at the core of TestRNN. In particular, there are two types of mutation function m in Algorithm 1: random mutation m_{rnd} and targeted mutation m_{targ}.

1) Random Mutation: When the LSTM input has continuous values (e.g., image input), Gaussian noises with fixed mean and variance are added to the input. Meanwhile, for discrete value input (e.g., IMDB movie reviews for sentiment analysis), a set of problem-specific mutation functions $\mathcal{M}$ are defined. The detail is in the experiment setup (see Section VII-A).

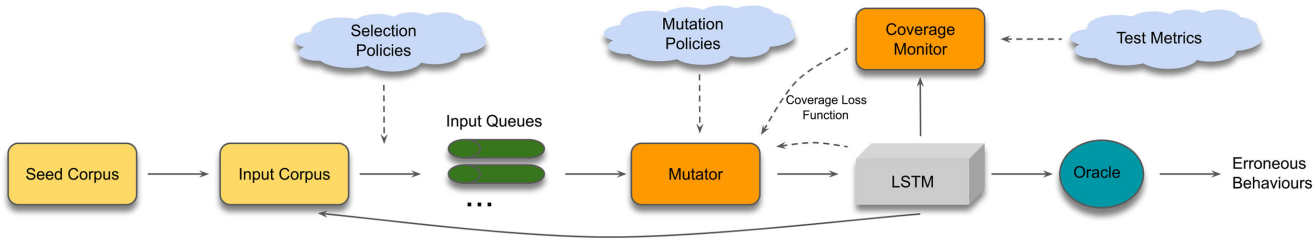

Fig. 4. Coverage-guided LSTM testing in TESTRNN.

Algorithm 1: TESTRNN Algorithm.

Input:
ϕ: RNN to be tested
$\mathcal{T}_0$: a set of seed inputs
m: a mutation function
r_{oracle}: oracle radius
Output:
$\mathcal{T}$: a set of test cases
$\mathcal{T}_{adv}$: a set of discovered adversarial samples

$\mathcal{T} \leftarrow \mathcal{T}_0$
$orig \leftarrow dict()$
$orig[x] \leftarrow x$ for all $x \in \mathcal{T}_0$
while *coverage rate is not satisfied* **do**
  $x \leftarrow$ select an element from $\mathcal{T}$
  $x' \leftarrow m(x)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{x'\}$
  $orig[x'] \leftarrow orig[x]$
  if $||orig(x) - x'||_2 \leq r_{oracle}$ *and* $\phi(x) \neq \phi(x')$ **then**
    $\mathcal{T}_{adv} \leftarrow \mathcal{T}_{adv} \cup \{x'\}$
return $\mathcal{T}, \mathcal{T}_{adv}$

2) Targeted Mutation: The targeted mutation is based on genetic algorithm for test case generation. Genetic algorithm is an evolutionary approach inspired by the process of natural selection. Mutations are selected only when they improve over the existing test cases on some predefined fitness function. The implementation of genetic algorithm comprises four steps: initialization, selection, crossover, and mutation. The last three steps are running iteratively till the solution is found.

a) Initialization: First, we initialize the population by choosing a test case from the previous running cases. The test case is very close to the satisfaction of test condition.

b) Selection: Next, we select best few test cases from the population to the mating pool, evaluated by the fitness function. For the three classes of test conditions (BC, SC, TC) with respect to some $s \in \mathcal{S}$ and $a \in \mathcal{A}$, we define the following fitness function as the distance to their respective targets, e.g.,

$$J_{\text{BC}}(x) = \alpha_{\max} - \mathcal{N}_m(\xi^{s,a}_{x,t})$$
$$J_{\text{SC}}(x) = \alpha_{\text{SC}} - \mathcal{N}_m(\Delta\xi^{s}_{x,t})$$
$$J_{\text{TC}}(x) = \sum_{j=1}^{w} \text{dist}(\mathcal{N}_z(\hat{\xi}^{s}_{x,j}), u_j)$$

where t, t_1, t_2, and j are time steps that can be inferred from the context. $u_j = [u_l, u_r]$ is the interval of subrange, represented by some symbol in Γ. The fitness of temporal curve to the targeted symbolic curve is to calculate the Manhattan distance, the absolute difference between structure value $\mathcal{N}_z(\hat{\xi}^{s}_{x,j})$, and the symbolic interval u_j is

$$\text{dist}(\mathcal{N}_z(\hat{\xi}^{s}_{x,j}), u_j) = \begin{cases} \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) - u_r & \text{if } \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) > u_r \\ u_l - \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) & \text{if } \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) < u_l. \\ 0 & \text{else} \end{cases}$$

Intuitively, the fitness function (also called coverage loss) $J(x)$ estimates the distance to the satisfaction of an unfulfilled test condition. $J(x) \leq 0$ means that the test condition is covered. By generating test cases with the objective of gradually minimizing the loss, the targeted mutation is essentially a greedy search algorithm.

Example 6.1: In Fig. 3, the symbolic representation of temporal curve is *acbab*; the fitness of the curve to test condition *bbcca* can be calculated as $J_{\text{TC}} = J_1 + J_2 + J_3 + J_4 + J_5$.

c) Crossover: Crossover is trivial in our test case generation for RNNs. This is mainly because the inputs to RNNs are usually discrete, which means the offspring of two parents by crossover (such as exchanging chromosome) may be invalid due to the undefined semantic meanings. To avoid the validity issue, we skip this step and using mutation methods directly.

d) Mutation: We randomly mutate the test cases in the mating pool with user-defined function $\mathcal{M}$ in order to generate a new population for the next iteration. The previous population is replaced with the new one. It should be noticed that the parents in the mating pool are also added to the new population to make sure that solutions are toward good directions during the iterations.

The targeted mutation m_{targ} is shown in Algorithm 2. At each iteration, we choose k (or $|P|$, whichever is smaller) best individuals in the population P. Each individual is utilized to generate n test cases via the random mutation function m_{rnd}. The old population P will be replaced with the mutants along with their k parents in P^*. The whole process is repeated until the test condition is met or the maximum iteration is exceeded.

C. Test Set Evaluation

1) Test Oracle: Test oracle determines if a test case passes or fails. We define a set of norm-balls, each of which is centered around a data sample with known label. The radius r_{oracle} of norm-balls intuitively means that a human cannot differentiate between inputs within a norm ball. In this article, Euclidean

Algorithm 2: Targeted Mutation.

Input:
J: fitness function
γ: maximun iteration number
k: number of parents for mating pool
n: number of offsprings mutated from one parent
m_{rnd}: a random mutation function
x': a test case that is the closest to satisfy test condition
Output:
x'_{new}: a test case covering new test condition

$itr \leftarrow 0$
$P \leftarrow \{x'\}$
while *test condition is not satisfied in* P *and* $itr < \gamma$ **do**
  sort individual $x \in P$ according to fitness $J(x)$
  $P^* \leftarrow$ highest sorted $\min\{k, |P|\}$ individuals in P
  $P \leftarrow P^* \cup m_{rnd}(P^*, n)$
  $itr \leftarrow itr + 1$
$x'_{new} \leftarrow \text{argmin}_{x \in P} J(x)$
return x'_{new}

distance, i.e., L^2-norm $||\cdot||_2$ is used. A test case x' is said to not pass the oracle if x' is within the norm-ball of some known sample x, i.e., $||x - x'||_2 \leq r_{\text{oracle}}$, and if x' has a different classification from x, i.e., $\varphi(x) \neq \varphi(x')$. Take the definition in [42], a test case does not pass the oracle is an adversarial sample. We use *adversary rate* to denote the percentage of test cases that do not pass the oracle.

2) Diversity of Test Set: More diversified test cases will explore more input space and, thus, are more likely to uncover different defects. Unfortunately, a unified, accurate way to measure diversity may not exist. We consider the following three intuitive, yet measurable proxies to the diversity.

First, diversity can refer to the number of categories the generated test cases belonging to. Intuitively, if the labels of two test cases are different, they are dissimilar and more diversified than two test cases with the same label. Second, test metrics (e.g., NC, SC, BC, and TC) are to guide the exploitation of different internal behaviors of RNNs (cf., Section IV). Therefore, a test set that can achieve higher coverage on more test metrics is more diversified than the other. Third, if the distance in input space, measured with L_1, L_2, and L_∞ norm, can represent the semantic similarity between test cases, we may define the diversity by quantifying the relative positions of test cases to the seed input. Suppose a test set $\mathcal{T}$ contains n test cases, generated from a seed x_0, the angular-based diversity measure [15] of $\mathcal{T}$ is

$$\text{Diversity}(\mathcal{T}) = -\left(\sum_{i,j=1}^{n} \frac{< x_i - x_0, x_j - x_0 >}{||x_i - x_0||||x_j - x_0||}\right)/n^2. \quad (6)$$

This diversity measure is formed by the cosine similarity [52] and bounded by $[-1, 1]$. Since the test cases are generated by adding small perturbations to the seed input, the angular-based diversity is to measure if the test cases are uniformly distributed around the seed input x_0. A larger Diversity$(\mathcal{T})$ represents a more diversified $\mathcal{T}$.

VII. Evaluation

We evaluate our TESTRNN approach with an extensive set of experiments from the following aspects.

1) The diversity of test cases generated under the guidance of the coverage metrics (see Section VII-B).
2) The ability of detecting RNN defects (see Section VII-C).
3) The effectiveness of the test case generation algorithms (see Section VII-D).
4) The advantages over state-of-the-art attack tool [31] (see Section VII-E).
5) The difference from state-of-the-art RNN testing tool DeepStellar [12] (see Section VII-F).
6) The exhibition of LSTM internal working mechanism (see Section VII-G).

Specifically, we study the following research questions (RQs).

RQ1: Will the exploitation of internal behavior lead to the testing of different LSTM functions?
RQ2: Are our new metrics needed when we already have existing metrics?
RQ3: Will the exploitation of internal behavior lead to the detection of adversarial samples?
RQ4: Will the exploitation of internal behavior lead to the identification of backdoor attacks?
RQ5: Can the test case generation algorithm achieve high coverage for the proposed test metrics?
RQ6: What are the advantages of TESTRNN over attack-based methods for detecting adversarial samples?
RQ7: What are the similarities and differences between DeepStellar and TESTRNN?
RQ8: Are the testing results based on the proposed test metrics helpful on making LSTM interpretable?

All the experiments are run on a desktop with Intel(R) Core(TM) i7 CPU @ 3.80 GHz and 16-GB Memory.

A. Experimental Setup

1) RNNs Under Evaluation: Our experiments are conducted on a diverse set of LSTM benchmarks, including the following.

a) MNIST handwritten digits analysis by LSTM: The MNIST database, containing a set of 60 000 grey-scale images of size 28×28, is used to train an RNN model with four layers. The first two layers are LSTM layers, which are correspondingly connected and fed with rows of input images. That is, each input image is encoded as the row vector of shape (28, 128) by the first LSTM layer, and then second layer will do further processing to output an image vector representing the whole image. Finally, two fully connected layers with ReLU and SoftMax activation functions, respectively, are used to process the extracted feature information to get the classification result. The model achieves 99.2% accuracy in training dataset (50 000 samples) and 98.7% accuracy in the default MNIST test dataset (10 000 samples).

b) Sentiment analysis by LSTM: The sentiment analysis network has three layers, i.e., an embedding layer, an LSTM

layer, and a fully connected layer, with 213 301 trainable parameters. The embedding layer takes as input a vector of length 500 and outputs a 500×32 matrix, which is then fed into the LSTM layer. Subsequently, there is a fully connected layer of 100 neurons.

c) Lipophilicity analysis by LSTM: We trained an LSTM regression network on a Lipophilicity dataset from the MoleculeNet [50]. The model has four layers: an embedding layer, an LSTM layer, a dropout layer, and a fully connected layer. The input is a SMILES string representing a molecular structure and the output is its prediction of Lipophilicity. A dictionary is used to map the symbols in the SMILES string to integers. We use the length of the longest SMILES in training dataset as the number of cells for the LSTM layer. Similar to text processing in the IMDB model, short SMILES inputs are padded with 0 s to the left side. We use the root-mean-square error (RMSE) as the measurement of model accuracy. Our trained model achieves RMSE $= 0.2371$ in training dataset and RMSE $= 0.6278$ in test dataset, which are better than the traditional and convolutional methods used in [50].

d) Video recognition for human behavior: A large-scale VGG16+LSTM network is trained over the UCF101 dataset [39]. VGG16, a CNN for ImageNet, extracts features from individual frames of a video. Then, the sequence of frame features are analyzed by LSTM layer for classification.

2) Test Metrics: We conduct experiments on several concrete test metrics, i.e., BC (for $\xi_t^{f,\text{avg}}$), SC (for $\Delta\xi_t^h$), and TC (for ξ_t^h). The configuration of thresholds are presented in Table I. Although the proposed three test metrics can be applied to every internal vector of LSTM cell, such as f, i, o, c, h, the current settings represent better semantic meanings. The interpretation of the testing results is discussed in Section VII-G.

3) Input Mutation: For MNIST model, we add Gaussian noise to input image and round off the decimals around 0 and 1 to make the pixel value stay within the value range.

The input to IMDB model is a sequence of words, on which a random change may lead to an unrecognizable (and invalid) text paragraph. To avoid this, we take a set $\mathcal{M}$ of mutants from the EDA toolkit [48], which was originally designed to augment the training data for improvement on text classification tasks. This ensures the mutated text paragraphs are always valid. In our experiments, we consider four mutation operations, i.e., $\mathcal{M}$ includes synonym replacement, random insertion, random swap, and random deletion. The text meanings are reserved in all mutations. For Lipophilicity model, we take a set $\mathcal{M}$ of mutants that change the SMILES string without affecting the molecular structure it represents. The enumeration of possible SMILES for a molecule is implemented with the Python cheminformatics package RDkit [35]. Each input SMILES string is converted into its molfile format, based on which the atom order is changed randomly before converting back. There may be several SMILES strings representing the same molecular structure. The enumerated SMILES strings are the test cases.

For UCF101 model, we add Gaussian noise to the original video frames instead of the feature inputs to the LSTM layer.

4) Oracle Setting: We use one fixed oracle radius for each model across all experiments. For continuous inputs, such as images and videos, we calculate the Euclidean distance as the measurement of perturbation. For the discrete inputs, such as text, We refer to the alpha parameter provided by the EDA toolkit, which approximately means the percent of words in the sentence that will be changed. That said, r_{oracle} for each RNN are listed in Table II. Note, we let $r_{\text{oracle}} =$ None for the Lipophilicity model, suggesting that no constraint is imposed on the norm ball. Hence, the determination of adversarial example is completely based on the classification. This is because, as suggested before, the test cases are only generated from those SMILES strings with the same molecular structure.

TABLE I
SUMMARY OF RNN MODELS UNDER TESTING

Test Model	No. of Classes	Test Acc.	Seq. of Interest	oracle
MNIST	9	98.7%	[4,24]	0.01
IMDB	2	86.2%	[400,500]	0.05
Lipophilicity	None	RMSE = 0.6278	[60,80]	None
UCF101	101	88.6%	[1,11]	0.1

TABLE II
CONFIGURATION OF TEST METRICS

Coverage Metrcis	Parameter Configuration
Neuron Coverage (NC)	Threshold = 0
K-multisection Neuron Coverage (KMNC)	$k = 10$
Neuron Boundary Coverage (NBC)	LB = -0.7, UB = 0.7
Strong Neuron Activation Coverage (SNAC)	UB = 0.7
Boundary Coverage (BC)	$\alpha_{max} = 0.8$
Step-wise Coverage (SC)	$\alpha_{sc} = 0.6$
Temporal Coverage (TC)	$w = 5$, $\lvert\Gamma\rvert = 3$

TABLE III
IMPACT OF SEEDS TO COVERAGE METRICS

Test Model	Seeds Input & Test Cases	Categories of Seeds	Coverage Metrics						
			NC	KMNC	NBC	SNAC	BC	SC	TC
MNIST	100 / 5000	1	0.93	0.65	0.41	0.44	0.10	0.43	0.38
		10	1.00	0.88	0.77	0.80	0.95	0.86	0.79
IMDB	100 / 5000	1	1.00	0.29	0.01	0.01	0.23	0.45	0.24
		2	1.00	0.37	0.01	0.01	0.81	0.54	0.64
Lipophilicity	10 / 2000	1	0.81	0.31	0.05	0.06	0.00	0.00	0.04
		10	1.00	0.86	0.68	0.66	0.95	0.95	0.90
UCF101	100 / 5000	1	1.00	0.47	0.07	0.06	0.00	0.00	0.16
		10	1.00	0.76	0.36	0.34	0.58	0.58	0.67

Bold entities are larger numbers through each comparison

B. Diversity of Test Cases

Test metrics can be seen as a proxy to exploit the input space, and intuitively more diversified test cases will explore more input space and, thus, are more likely to uncover different defects. Thus, we investigate if the achievement of high coverage will indeed lead to the testing of different LSTM functions (*RQ1*), and if our new metrics encourage the exploitation of more regions in the input space than existing metrics (*RQ2*).

1) Approximation of LSTM Functional Coverage (RQ1): Table III shows that LSTM model's functional coverage can be approximated by using TESTRNN metrics. This is based on the assumption that a data label (i.e., category) corresponds to a "functional feature" of the LSTM. We observe that by only using one category of seeds input, it is hard to achieve high coverage rate for TESTRNN metrics, even when thousands of test cases are generated. In contrast, with seeds input from more categories, the generated test cases from targeted mutation can broadly explore the input space and more internal behaviors of RNNs. Thus, all

TABLE IV
COMPLEMENTARITY OF TEST METRICS: COMPARISON BETWEEN NEURON LEVEL TEST METRICS AND THE PROPOSED TESTRNN METRICS IN MINIMAL TEST SUITE

Test Model	Target Metrics	Neuron Level Metrics				TESTRNN Metrics		
		NC	KMNC	NBC	SNAC	BC	SC	TC
MNIST	NC	1.00	0.61	0.39	0.44	0.10	0.00	0.10
	KMNC	1.00	0.85	0.67	0.72	0.10	0.29	0.44
	NBC	1.00	0.77	0.73	0.75	0.14	0.19	0.28
	SNAC	0.98	0.73	0.62	0.75	0.10	0.14	0.19
IMDB	NC	1.00	0.23	0.00	0.00	0.00	0.00	0.02
	KMNC	1.00	0.39	0.01	0.01	0.01	0.02	0.07
	NBC	0.48	0.13	0.01	0.01	0.00	0.00	0.01
	SNAC	0.47	0.10	0.01	0.01	0.00	0.00	0.00
Lipophilicity	NC	1.00	0.43	0.16	0.16	0.05	0.05	0.03
	KMNC	1.00	0.92	0.70	0.69	0.40	0.20	0.38
	NBC	1.00	0.84	0.81	0.78	0.50	0.20	0.22
	SNAC	1.00	0.77	0.62	0.78	0.40	0.20	0.13
UCF101	NC	1.00	0.48	0.26	0.36	0.10	0.10	0.15
	KMNC	1.00	0.74	0.60	0.66	0.18	0.10	0.20
	NBC	1.00	0.65	0.75	0.58	0.15	0.18	0.16
	SNAC	1.00	0.76	0.68	0.84	0.22	0.18	0.20

rates of TESTRNN coverage metrics are significantly improved, given the test set. Table III also records the coverage of neuron level metrics, which are widely used in the CNNs/FNNs. These test metrics show less sensitivity with respect to the diversity of functional features in the test suite, e.g., the NC coverage can already reach almost 100% by only using test cases of one label.

Answer to RQ1: The exploitation of internal behavior by TESTRNN can approximate the testing of different LSTM functional features.

2) Comparison With Neuron Level Coverage (RQ2): We implement the NC [33], k-multisection NC (KMNC), neuron BC (NBC), and strong neuron activation coverage (SNAC) [25] on the testing layers of our LSTM models. We note that the concept "neuron" is ambiguous in RNNs, since the hidden output of RNNs' cells are vectors. Here, we consider covering each element of the hidden output h in the testing layer. Results are presented in Tables III and IV.

In the experiments, we find that NC can be trivially achieved. Shown in Table III, NC is not suitable for exploring RNNs' internal functionality, since one category's seeds input is enough for the high coverage of neuron activation. Moreover, we find that other neuron level test metrics may be impossible to satisfy for IMDB test model. The low coverage rate of KMNC, NBC, and SNAC indicates that the activation of neurons for IMDB model is concentrated in a small interval. In other words, the neuron level test metrics cannot be a good option to search for diverse test cases.

Table IV shows the complementarity of neuron level test metrics and our proposed TESTRNN metrics. A set of complementary test metrics (and test cases) can enhance the diversity of the testing. In the experiments, we take *minimal test suite*, in which the removal of any test case may lead to the reduction of coverage rate. The consideration of the minimality of test suite enables a fair comparison since it reduces the overlaps as much as possible. The results confirm that a test suite that can achieve high coverage for neuron level test metrics is not necessary to get the high coverage for RNN test metrics. For example, in the MNIST LSTM model, test cases that achieve 100% NC can only cover less than 10% of the overall test conditions by TESTRNN metrics (with 10% BC, 0% SC, and 10% TC). Similar patterns also happen to other models. That means the proposed test metrics provide the guide for the selection of additional test cases, which are complementary to those guided by the neuron level test metrics.

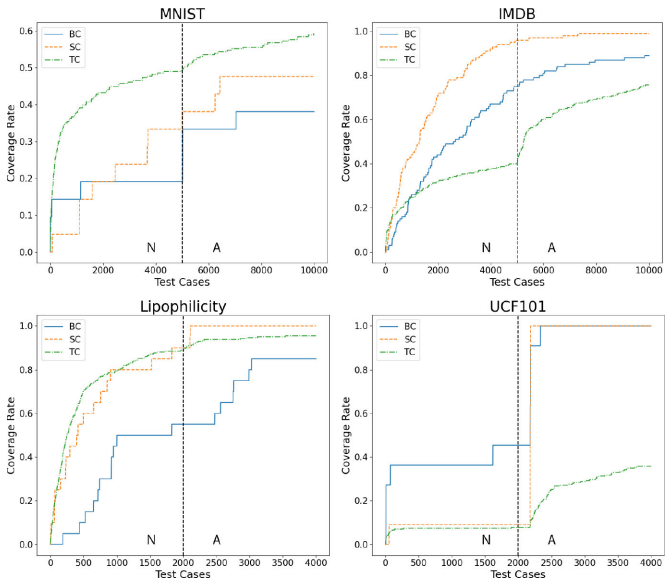

Fig. 5. Update of coverage with normal perturbed samples ("*N*") and adversarial samples ("*A*").

Moreover, we discover that there are many redundancy of test requirements, regarding to the relation between individual test metric in neuron level category. For example, if we derive a test suite that targets at increasing the coverage of KMNC, 'NBC, and SNAC, both' get the high coverage results.

Answer to RQ2: The TESTRNN metrics exhibit a dramatic portion of LSTM internal behaviors that cannot be explored by existing metrics.

C. Detecting RNN Defects

1) Searching for Adversarial Samples (RQ3): We collect the set of normal perturbed samples (*N*) and adversarial samples (*A*), respectively. First, normal perturbed samples are added to the test set to witness the increase of the coverage. When the coverage is difficult to improve, adversarial samples are considered. The update of whole process is illustrated in Fig. 5. The dashed vertical line distinguish the coverage update with normal perturbed samples from that with adversarial samples. It should be noted that the coverage update of some test metrics is stepped growth, due to the small amount of total test conditions, which is shown in Table II.

Fig. 5 reveals that normal perturbed samples can only satisfy part of test conditions, whereas the rest are more sensitive to the adversarial samples. In all the plots, coverage of RNN test metrics can be further increased in consideration of adversarial samples. A more obvious example is, the TC coverage of IMDB model tend to saturate in the left side when only normal perturbed samples are utilized. In the right side, the coverage curve becomes steep, indicating the discovery of test cases capturing new internal behaviors.

TABLE V
COMPARING THE ROBUSTNESS OF MODELS VIA COVERAGE-GUIDED TESTING

Test Dataset	Model No.	Test Cases	Adv. Samples Rate	Unique Adv. Samples	Coverage Metrics BC	SC	TC
MNIST	1	5958	0.060	176	0.48	0.81	0.90
	2	3570	0.075	184	0.57	0.86	0.90
IMDB	1	5841	0.039	138	0.94	0.93	0.75
	2	1575	0.047	68	0.62	0.72	0.75
Lipophilicity	1	2936	0.371	191	0.95	1.00	0.95
	2	6727	0.010	44	0.88	1.00	0.95
UCF101	1	6352	0.420	182	0.98	0.95	0.60
	2	6100	0.250	90	0.95	0.92	0.60

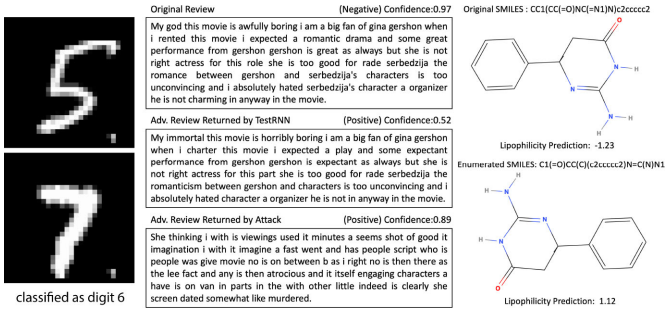

Fig. 6. Backdoor samples for MNIST model (left). Adversarial samples for IMDB (middle) and Lipophilicity (right) models.

In addition to the sensitivity of test metrics to adversarial samples, we show how to compare the robustness of models via coverage-guided testing. We use TC as the termination condition to generate a test suite and calculate the adversarial rate. To achieve the high coverage of test metrics, we use genetic algorithm for test case generation, more details of which can be seen in Section VI-B. The other settings remain the same for the fair comparison.

As shown in Table V, adversarial samples rate and number of unique adversarial samples in the generated test suite are two important indicators for the robustness evaluation. The unique adversarial samples refer to the adversarial samples crafted from distinct seeds input. For a set of trained models, we pursue the model, the test suite of which contains less amount of adversarial samples and unique adversarial samples. For example, we pick up model 2 for Lipophilicity prediction, since the values of two indicators are way smaller than that of model 1. We comment that with large enough amount of test cases, coverage-guide testing approach provides a new way for the measure and selection of more robust classifier. This is compatible with the results in [40] that a poorly trained neural network exposes more adversarial samples subject to well-defined coverage-guided testing.

Answer to RQ3: By exploiting the model's internal behaviors, TESTRNN is able to capture the LSTM adversarial samples.

2) Detecting Backdoor Input in RNNs (RQ4): We investigate the possibility of applying coverage-guided testing to the detection of backdoor input in neural networks. We try to exploit if there is any difference between clean input and backdoor input, which can be captured by our proposed test metrics. Examples of backdoor input are illustrated in Fig. 6. We train two handwritten digits recognition models, one of which is benign classifier and the other one is the malicious classifier subject to the backdoor attack in [17]. Table VI shows that both benign and malicious

TABLE VI
SENSITIVITY OF TEST METRICS TO BACKDOOR SAMPLES IN MNIST DATASET

Model	Test Acc. (C / B)	Data	Class 0 BC	SC	TC	Class 6 BC	SC	TC	Class 9 BC	SC	TC
Benign	99.1% / 9.5%	T	0.39	0.25	0.16	0.29	0.18	0.30	0.32	0.29	0.22
		T + C	0.39	0.25	0.17	0.29	0.18	0.30	0.32	0.29	0.22
		T + B	0.39	0.25	0.17	0.29	0.18	0.30	0.32	0.29	0.22
Malicious	98.7% / 100%	T	0.39	0.18	0.30	0.25	0.18	0.60	0.07	0.18	0.27
		T + C	0.39	0.18	0.30	0.25	0.18	0.60	0.07	0.18	0.27
		T + B	0.39	0.25	0.33	0.25	0.21	0.63	0.07	0.21	0.29

classifiers keep good prediction performance in clean test set. For the backdoor test set, benign classifier keep the normal accuracy, whereas the malicious classifier predicts inputs with the backdoor trigger as the attacked label successfully.

We conduct sensitivity analysis by computing the coverage of the proposed test metrics in training data (T), clean test data (C), and backdoor test data (B) for each classifier. In the first row of Table VI, we calculate the coverage of the training data from same class. On the basis of this, we add clean test data or backdoor test data for evaluation. If the coverage rate is further increased in second and third rows, the new internal patterns are discovered. The experimental results describe that backdoor input activate same internal behavior with clean input for a benign classifier. In contrast to this, the backdoor input to malicious classifier will induce different internal activation, which can be seen from the apparent increase of coverage in T+B. Although the backdoor input is very similar to the clean input with a small region of pixels changed (see Fig. 6), the internal activation in the malicious model can still be revealed by the coverage change of the proposed TESTRNN metrics.

We remark that the aforementioned experiment only confirms that test metrics are sensitive to backdoor samples when testing an attacked model. More accurate detection of backdoor in RNNs needs more precise refinement of test metrics, e.g., adding the backdoor knowledge to the metrics design on top of the structure information. Nevertheless, the goal of coverage-guided testing is still diversifying the test suite so that defects, such as backdoor samples, are more likely to be detected.

Answer to RQ4: The TESTRNN metrics can identify the difference between the backdoor input and the normal input (to malicious models).

D. Effectiveness of Test Case Generation (RQ5)

We show the effectiveness of our test case generation from the following aspects: (1) it is nontrivial to achieve high coverage rate, and (2) there is a significant percentage of adversarial samples in the generated test suite. For (1), we show that the targeted mutation (i.e., random mutation enhanced by genetic algorithm) is needed to boost the coverage rate. Three test case generation methods are considered: (Seeds) sampling 200 seeds input from training dataset, (Ran.) generating test cases from seeds by using random mutation, and (Targ.) generating test cases from seeds by using targeted mutation. Fig. 6 demonstrates detected adversarial samples for IMDB and Lipophilicity models, and we omit other models for brevity. All experimental results are based on five runs with different random seeds. The results are averaged and summarized in Table VII. For each test case generation method and LSTM model, we also report the

TABLE VII
EXPERIMENTS FOR TEST CASE GENERATION METHODS

Test Model	Test Gen. Method	Test Cases	No. of Adv. samples	Avg. Perturb.	Unique Adv. Samples	Coverage Metrics BC	SC	TC
MNIST	Seeds	200	-	-	-	0.43	0.14	0.34
	Ran.	10000	226	1.180	18	0.57	0.52	0.66
	Targ.	10000	244	1.497	32	1.00	1.00	0.79
IMDB	Seeds	200	-	-	-	0.11	0.05	0.24
	Ran.	10000	308	0.136	88	0.84	0.40	0.77
	Targ.	10000	367	0.103	97	1.00	0.58	0.82
Lipophilicity	Seeds	200	-	-	-	0.65	0.55	0.48
	Ran.	2000	812	-	190	0.95	1.00	0.91
	Targ.	2000	834	-	194	1.00	1.00	0.95
UCF101	Seeds	200	-	-	-	0.52	0.53	0.11
	Ran.	10000	3613	1.031	112	0.82	0.90	0.31
	Targ.	10000	4201	1.251	156	1.00	1.00	0.66

TABLE VIII
COMPARISON BETWEEN DEEPSTELLAR AND TESTRNN USING MNIST: 100 000 TEST CASES ARE GENERATED FROM 100 SEEDS

DeepStellar				TestRNN			
Test Metrics	Seeds	S-Guid.	T-Guid.	Test Metrics	Seeds	Ran.	Targ.
BS	0.45	0.80	0.82	BC	0.14	0.57	1.00
BT	0.11	0.32	0.63	SC	0.38	0.67	1.00
WT	0.76	0.90	0.95	TC	0.24	0.70	1.00
Adv. Samples	-	1588	1661	Adv. Samples	-	1778	1830

number of adversarial samples, unique adversarial samples in the test suite, and their average perturbation. This experiment considers all four models.

Table VII shows that the coverage rates and the number of adversarial samples for Ran. are significantly higher than those of Seeds, that is, Ran. is effective in finding the adversarial samples around the original seeds. Furthermore, if we use Targ., both the coverage rates and the number of adversarial samples are further increased. The aforementioned observations confirm the following two points: our test metrics come with a strong bug finding ability; and higher coverage rates indicate more comprehensive test. We remark that the TC rates for UCF101 model are relatively low and harder to improve, because the mutations are made on the image frames (i.e., before CNN layers) instead of directly on the LSTM input. This shows that the adversarial samples for CNNs are orthogonal to those of LSTMs, another evidence showing that test metrics for CNNs cannot be directly applied to RNNs.

Answer to RQ5: The test case generation algorithm is effective in improving both the coverage rate and the adversary rate. In particular, the targeted mutation method can be utilized to find more corner samples.

E. Comparison With Attack-Based Defect Detection (RQ6)

We compare TESTRNN with state-of-the-art RNN adversarial attack [2], [31], which detects robustness defects. These attack algorithms utilize the model's gradient over input sequence to iteratively change some parts of the input that contribute the most to the model's prediction. Their methods can successfully find adversarial samples. However, these attack methods have two main drawbacks, when compared with our testing method.

First, attack methods search for adversarial samples by adding perturbations in the gradient direction. This easily leads to the situation where the generated adversarial samples are concentrated in a "buggy" area of the input space, as shown in Table IX and Fig. 7. We first collect the same amount of adversarial samples in MNIST model returned by attack methods and TESTRNN, respectively. Then, we calculate the angular-based

TABLE IX
ANGULAR-BASED DIVERSITY (A GREATER VALUE REPRESENTS A BETTER DIVERSITY) AND AVERAGE PERTURBATION (SMALLER IS BETTER) OF ADVERSARIAL SAMPLES

Seed	Angular-based Diversity TESTRNN	DeepStellar	Attack	Avg. Perturb. TESTRNN	DeepStellar	Attack
1	-0.277	-0.468	-0.598	0.006	0.012	0.014
2	-0.289	-0.438	-0.556	0.006	0.010	0.013

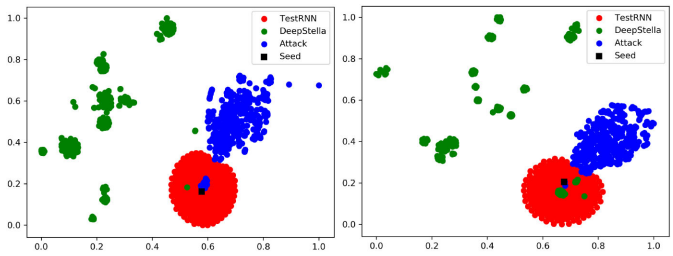

Fig. 7. Visualization of adversarial samples generated by TESTRNN, DeepStellar, and gradient-based attack, respectively, in MNIST model. The visualization is conducted by projecting high-dimensional images onto a two-dimensional space. Each figure corresponds to a seed input in the dataset.

diversity of each set (see Table IX) and apply the principal component analysis (PCA), a well-known dimensionality reduction technique, to project the high-dimensional adversarial images onto two-dimensional space for better visualization (see Fig. 7). We can see from the resulting diversity measurement and visualization that compared to attack methods, our testing method exercises different behaviors of RNN and generates a diverse set of test cases, intensively covering the input region around the seed input. *This ability will be helpful in exposing more types of defects of the RNN* (not merely in the gradient direction).

Moreover, RNNs are widely applied to the nature language processing, in which the inputs to an RNN, i.e., words, are discretely distributed. Attack methods aggressively replace important words in the text and produce an adversarial sequence. In this process, it is hard to consider both the gradient and the whole text's semantic meaning. That is, the modified text may easily become human-unreadable and impossible to occur in real world. On the other hand, our testing method is able to reduce such problems by taking the mutants from off-the-shelf tools, such as the EDA toolkit. Fig. 6 presents adversarial movie reviews returned by attack method and TESTRNN, respectively. It is easy to see that the adversarial review returned by the gradient attack is hard to comprehend, whereas the one from TESTRNN is much easier.

Answer to RQ6: The TESTRNN is able to generate a set of diverse and natural test cases, so as to expose more types of defects.

F. Comparison With State-of-the-Art Testing Methods (RQ7)

We compare TESTRNN with DeepStellar, a state-of-the-art testing tool dedicated for RNNs. As discussed in Section V-B, two different test metrics are integrated in DeepStellar, i.e., state coverage and transition coverage, which are corresponding to BC and SC in TESTRNN. Apart from these, TESTRNN have TC

TABLE X
COMPLEMENTARITY OF TEST METRICS IN DEEPSTELLAR AND TEST RNN

Tool	Test Gen. Method	Target Metrics	Coverage Metrics					
			BS	BT	WT	BC	SC	TC
TestRNN	Ran.	-	0.78	0.63	0.94	0.57	0.67	0.70
	Targ.	BC,SC,TC	0.78	0.64	0.95	1.00	1.00	1.00
DeepStellar	S-Guide.	BS	0.80	0.32	0.90	0.05	0.10	0.12
	T-Guide.	BT,WT	0.82	0.63	0.95	0.10	0.24	0.20

for the internal sequential processing behavior of RNNs. We start from 100 seeds drawn from training set and generate 100 000 test cases by DeepStellar and TESTRNN, respectively. The test suites are evaluated for the coverage rate and number of adversarial samples. We compute BS, BT, and weighted transition coverage (WT) in DeepStellar guided by different generation strategy, S-Guide, and T-Guide. The testing results for both are recorded in Table VIII. First, we can see that test metrics in DeepStellar already have high coverage rates upon seeds input, as opposed to our metrics, which display relatively smaller coverage rates upon the same seeds. That means that our metrics are better for exploiting the input space around seeds. Second, DeepStellar adopts the fuzzing strategy with the guidance of different test metrics, which is effective to boost the coverage. However, in this experiment for small-scale model trained on MNIST, 100 000 test cases are still not enough for 100% coverage of test requirements in DeepStellar. It seems that some of their defined test requirements may be infeasible to satisfy. On the contrary, TESTRNN can achieve a relatively high coverage results with random mutation and the coverage rates of all the metrics can be significantly boost to achieve 100% by genetic algorithm based mutation method. The number of adversarial samples in the test suite reflects that TESTRNN is superior to DeepStellar in terms of exploiting diverse internal behaviors and bugs finding ability.

To understand the relative merits of the defects returned by DeepStellar, TESTRNN, and gradient-based attack, respectively, we compute the angular-based diversity and average perturbation of each set (see Table IX), and visualize them with PCA projection (see Fig. 7). We can see that the adversarial samples from DeepStellar are sparsely distributed and most of them are more distant to the seed input. TESTRNN explores space that is close to the seed input. This aligns better to the goal of adversarial testing, which is to find more bugs around the seed with as small perturbations as possible (the bugs are more realistic/natural, thus more likely to exist in real world).

In addition to the comparison of testing results, we are also interested in the complementarity of test metrics in TESTRNN and DeepStellar. We derive the minimal test suites by different test case generation methods. Then, the test suite generated by TESTRNN is evaluated for the coverage of metrics in DeepStellar, and vice versa. Results in Table X suggests that test suite generated by TESTRNN can easily achieve high coverage rate for the metrics in DeepStellar. We find that test suite produced by DeepStellar cannot get high coverage rate on our metrics. This confirms our discussion of the relation between coverage metrics in Section V-B.

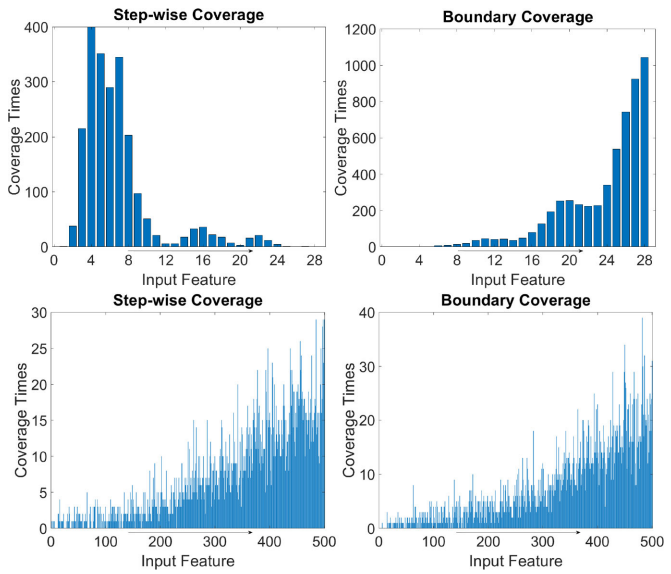

Fig. 8. 2000 test cases are used to demonstrate the coverage times of 28 features in an LSTM layer of MNIST model (first line) and 500 input features in LSTM layer of IMDB sentiment analysis model (second line).

Answer to RQ7: The TESTRNN test generation can achieve high coverage of the test metrics in DeepStellar, but not vice versa.

G. Exhibition of Internal Working Mechanism (RQ8)

In this section, we show that the working mechanism of LSTM networks can be understood via the test cases generated from TESTRNN. We conduct experiments to visualize the learning process of LSTM layer via TESTRNN results.

Coverage times denote the number of times a test condition is satisfied by running the test suite. Intuitively, coverage times represent the level of difficulty of asserting an input feature. Fig. 8 reports the coverage times for each input feature. We note that in BC and SC, each input feature x_t corresponds to a test condition on $\xi_t^{s,a}$, as in MNIST, it is defined with respect to a row of pixels on the input image. In sentiment analysis model, the input feature refers to a word in movie reviews.

As discussed in Section IV, SC is to assert if an input feature is significant to the model prediction. Then, an important input feature will cause great changes of hidden memory h_t and satisfy the test condition of SC. BC monitors the forget gate values at each time step. The satisfaction of BC means the LSTM will not drop out the information stored in memory.

If we combine SC and BC plots, the whole working process of LSTM layer inside the MNIST model becomes transparent. The sequential input of an image starts from the top row and finishes at the bottom row. At the beginning, the top rows of MNIST images are blank and do not contribute to the model prediction. These less-important information is gradually thrown away from the memory. When the input rows containing digits are fed to the LSTM cells, the model will start learning and the short-term memory, represented by the outputs h_t, start to have strong reactions. When approaching the end of the input,

which corresponds to the bottom of the digit images, LSTM has already been confident about the final classification, and therefore, becomes lazy to update the memory. Overall, we can see that MNIST digits recognition is not a complicated task for the LSTM model and usually the top half of the images are sufficient for the classification.

For the IMDB model, the final classification is influenced by every input feature. To make sure that input features between 450–500 contain real words instead of padded 0 s, we take 2000 reviews whose lengths are greater than 50. We observe from the second line in Fig. 8 that the coverage times gradually increase, it might be the nature of test cases—most test cases contain text of length much less than 500. We therefore focus on the last 50 input features. We see that both BC and SC test conditions in the IMDB model are randomly activated, a phenomenon that is completely different from that of MNIST results. This can be explained as that the IMDB model does not have a fixed working pattern, such as the MNIST model. Sensitive words in a review may appear in any place of the text.

Answer to RQ8: The generated test suite can be used to understand the data processing mechanism of LSTM models. This is a step toward interpretable RNNs.

H. Threats to Validity

First, we fix the thresholds or symbols of test metrics for all the experiments. If we decrease the values of threshold (or reduce the symbols to represent subranges), the test conditions can be easier to satisfy, and fewer test cases are generated. Conversely, if we tighten the thresholds, more test cases are needed to cover the test conditions. The input space are more thoroughly explored.

Second, we only choose part of input sequence to test, details of which is shown in Table II. If we use TESTRNN to test the entire input sequence, some test conditions may be harder to meet. The choices of partial input sequences in our experiment are as follows. For MNIST dataset, the hand-written digits are usually concentrated on 4th to 24th rows out of 28 rows. The rest of the images are blank. For IMDB and Lipophilicity datasets, the input to RNNs are usually padded with 0 s. And the input 0 s only induce very small activation, which can be seen in Fig. 8.

We define unique mutation functions for different models to ensure the generated test input are always valid. Since we set thresholds of test metrics with reference to the training data, it is nontrivial to validate the test input. Mutation function needs to keep the semantics meanings of seeds input. For example, in the experiment of testing IMDB model, we mutate the text paragraph instead of the input to LSTM layer.

Some minor threats include the settings of oracle and random seeds. We also fix the configurations for these parameters to make all the experiments consistent. The oracle radius can affect the adversarial samples rate and the average perturbations in the test suite. If we set up a smaller oracle radius, the number of perturbed input recognized as adversarial samples and the average perturbations are both decreased. The random seeds are utilized to control the reproducibility of the experiments. In most experiments, we do several test with different seeds input and get the average results so that the accidental errors can be avoided.

VIII. Related Work

A. Adversarial Samples for RNNs

Since adversarial robustness is regarded as a major safety concern for deep learning [21], a number of works appear on generating adversarial samples for RNN tasks, such as natural language processing [31] and automated speech recognition [14]. In this article, we treat adversarial samples as a proxy to evaluate the effectiveness of the proposed coverage criteria.

B. Testing FNNs

Most neural network testing methods focus on FNNs. In [32], the NC is proposed for exploiting neuron activation conditions in an FNN. Various refinements and extensions of NC are later developed in [25]. Motivated by the usage of MC/DC coverage metrics in high-criticality software, in [40], a family of MC/DC variants are designed for FNNs, by taking into account the causal relation between features of different layers. Moreover, it has been shown in [40] that the criteria in [25] and [32] are special cases of the MC/DC variants.

In addition to the structural coverage criteria mentioned earlier, metrics in [7] and [49] define a set of test conditions to partition the input space. Though not being a coverage metric, the method in [22] measures the difference between training and test datasets based on structural information of FNNs.

Guided by the coverage metrics, test cases can be generated via various techniques, including, e.g., heuristic search [36], [54], fuzzing [18], [30], mutation [26], [46], and symbolic encoding [16], [41], etc. None of these works have considered RNNs. Refer to the work in [21] for a survey on techniques for the safety and trustworthiness of neural networks.

C. Testing RNNs

Few works contribute to the development of coverage metrics for RNNs. DeepStellar [12] first proposes to abstract an RNN model into a DTMC. The abstracted DTMC is an approximation, whose fidelity to the original RNN is unknown. Such approximation can lead to unexpected consequences for testing, including the false positives and false negatives due to the misplacement of faulty corner cases in RNN and DTMC. Moreover, only cell states c are utilized in the abstracted DTMC along with the development of test metrics. Other functional components of RNNs, including the gates f, i, o and the hidden output h, are not considered. As demonstrated in experiments, these components have their dedicated meanings and ought to be considered when a more extensive testing is expected. They are also helpful to improve the interpretability of testing results. Moreover, RNN-Test [19] develops some test metrics to work with the structural components (gate f, cell c, and output h) directly. Their test metrics can be viewed as special cases of our BC. More importantly, they do not study the temporal relations, which we believe are the most fundamental characteristics of RNNs (as opposed to CNNs). We think that the differences mentioned earlier are significant enough to distinguish the work of ours from that in [12] and [19].

D. Difference Between Testing and Defect Detection

Recent paper [51] on correlations between coverage criteria and model quality suggests that coverage-guided testing complements gradient-based adversarial attack. They discover that adversarial samples found by FNN coverage-guided testing can be further utilized to retrain more robust models. However, such models may not be robust to the gradient-based attack (e.g., PGD [28]). On the other hand, PGD-based adversarial training may improve models' robustness to the adversarial attack but not attacks with guidance of coverage metrics.

E. Visualization for LSTM

In each LSTM layer, a sequential input $\{x_t\}_{t=1}^n$ corresponds with a sequence of vectors for structural components, e.g., $\{f_t\}_{t=1}^n$ and $\{h_t\}_{t=1}^n$. These internal vectors are high dimensional and impossible to be comprehended by humans. Then, some dimensionality reduction methods (e.g., t-SNE and PCA) have been adopted to visualize the information behind them. For instances, Rauber *et al.* [34] employ PCA to extract the principle component of h_t at each time step t. These methods facilitate the interpretation of RNN's hidden behaviors. Dimensionality reduction methods have also been used to abstract a neural network into an abstracted model, such as a Bayesian network [3]. Our interpretation is completely different from the above, and works by visualizing the working process of LSTM layer based on a set of test cases.

F. Neural Network Repairing

The repairing of neural network has also been studied, with the aim to utilize the generated test cases to improve the model's adversarial robustness [27], [45], [53] or fix the detected backdoor [44]. In contrast to the typical machine learning retraining [47], such work often relies on properly designed test cases to first identify certain structures inside the neural network model that are responsible for the model's undesirable behaviors, and then correct the model's behavior by, e.g., retraining [27], weight adaption [53], symbolic constraint solving [44], etc. In [45], each test case's impact on improving the model robustness is quantified. Nevertheless, we are not aware of any repairing method that is designed for RNNs.

IX. Conclusion

In this article, we proposed a coverage-guided test framework for the verification and validation of RNNs. We developed a tool TESTRNN based on the test framework and validated it on several LSTM models, trained on popular benchmarks. In the future, we plan to investigate the possibility of utilizing the testing results to mitigate the RNN defects and also certify RNNs.

Acknowledgment

The authors would like to thank the DeepStellar team for sharing their code and setup that make the comparison of experiments possible. They would also like to thank the anonymous reviewers of this article for their inspiring comments that help strengthen this work.

References

[1] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 2890–2896.
[2] M. Behjati, S.-M. Moosavi-Dezfooli, M. S. Baghshah, and P. Frossard, "Universal adversarial attacks on text classifiers," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2019, pp. 7345–7349.
[3] N. Berthier, A. Alshareef, J. Sharp, S. Schewe, and X. Huang, "Abstraction and symbolic execution of deep neural networks with Bayesian approximation of hidden features," 2021, *arXiv:2103.03704*.
[4] P. Bishop and A. Povyakalo, "Deriving a frequentist conservative confidence bound for probability of failure per demand for systems with different operational and test profiles," *Rel. Eng. Syst. Saf.*, vol. 158, pp. 246–253, 2017.
[5] L. Brader, H. Hilliker, and A. C. Wills, *Testing for Continuous Delivery With Visual Studio 2012*. Redmond, WA, USA: Microsoft Press, 2012.
[6] B. Chen *et al.*, "Detecting backdoor attacks on deep neural networks by activation clustering," in *Proc. Workshop Artif. Intell. Saf. Co-Located 33rd AAAI Conf. Artif. Intell.*, 2019, vol. 2301, pp. 66–73.
[7] C.-H. Cheng, G. Nührenberg, C.-H. Huang, H. Ruess, and H. Yasuoka, "Towards dependability metrics for neural networks," in *Proc. 16th ACM-IEEE Int. Conf. Formal Methods Models Syst. Des.*, 2018, pp. 43–46.
[8] E. M. Clarke Jr., O. Grumberg, D. Kroening, *D. Peled, and H. Veith, Model Checking*. Cambridge, MA, USA: MIT Press, 2018.
[9] F. Crecchi, D. Bacciu, and B. Biggio, "Detecting black-box adversarial examples through nonlinear dimensionality reduction," in *Proc. 27th Eur. Symp. Artif. Neural Netw.*, 2019, pp. 483–488.
[10] J. Dai, C. Chen, and Y. Li, "A backdoor attack against LSTM-based text classification systems," *IEEE Access*, vol. 7, pp. 138872–138878, 2019.
[11] Y. Dong *et al.*, "There is limited correlation between coverage and robustness for deep neural networks," 2019, *arXiv:1911.05904*.
[12] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "DeepStellar: Model-based quantitative analysis of stateful deep learning systems," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 477–487.
[13] P. G. Frankl, R. G. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Trans. Softw. Eng.*, vol. 24,no. 8, pp. 586–601, Aug. 1998.
[14] Y. Gong and C. Poellabauer, "Crafting adversarial examples for speech paralinguistics applications," in *Proc. DYnamic Novel Adv. Mach. Learn. Intell. Cyber Secur. Workshop*, San Juan, PR, USA, 2018.
[15] Z. Gong, P. Zhong, and W. Hu, "Diversity in machine learning," *IEEE Access*, vol. 7, pp. 64323–64350, 2019.
[16] D. Gopinath, K. Wang, M. Zhang, C. S. Pasareanu, and S. Khurshid, "Symbolic execution for deep neural networks," 2018, *arXiv:1807.10439*.
[17] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
[18] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "DLFuzz: Differential fuzzing testing of deep learning systems," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 739–743.
[19] J. Guo, Y. Zhao, X. Han, Y. Jiang, and J. Sun, "RNN-Test: Adversarial testing framework for recurrent neural network systems," 2019, *arXiv:1911.06155*.
[20] R. G. Hamlet and R. Taylor, "Partition testing does not inspire confidence," *IEEE Trans. Softw. Eng.*, vol. 16, no. 12, pp. 1402–1411, Dec. 1990.
[21] X. Huang *et al.*, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Comput. Sci. Rev.*, vol. 37, 2020, Art. no. 100270.
[22] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, 2019, pp. 1039–1049.
[23] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural coverage criteria for neural networks could be misleading," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas Emerg. Results*, 2019, pp. 89–92.
[24] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," in *Proc. 2nd Workshop Temporal Data Mining*, 2002, pp. 53–68.
[25] L. Ma *et al.*, "DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," in *Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng.*, 2018, pp. 120–131.

[26] L. Ma *et al.*, "DeepMutation: Mutation testing of deep learning systems," in *Proc. IEEE 29th Int. Symp. Softw. Rel. Eng.*, 2018, pp. 100–111.
[27] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "MODE: Automated neural network model debugging via state differential analysis and input selection," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 175–186.
[28] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Representations*, 2018.
[29] Y. Ming *et al.*, "Understanding hidden memories of recurrent neural networks," in *Proc. 12th IEEE Conf. Vis. Analytics Sci. Technol.*, 2017, pp. 13–24.
[30] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "TensorFuzz: Debugging neural networks with coverage-guided fuzzing," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, vol. 97, pp. 4901–4911.
[31] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Proc. IEEE Mil. Commun. Conf.*, 2016, pp. 49–54.
[32] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 1–18.
[33] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," 2017, *arXiv:1712.01785*.
[34] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE Trans. Visualization Comput. Graph.*, vol. 23, no. 1, pp. 101–110, Jan. 2017.
[35] *RDKit: Open-Source Cheminformatics*. Accessed: Apr. 11, 2013. [Online]. Available: http://www.rdkit.org
[36] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 5944–5952.
[37] A. Shafahi *et al.*, "Poison frogs! Targeted clean-label poisoning attacks on neural networks," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6106–6116.
[38] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 3–18.
[39] K. Soomro, A. R. Zamir, and M.Shah, "UCF101: A dataset of 101 human action classes from videos in the wild," CRCV-TR-12-01, 2012.
[40] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Structural test coverage criteria for deep neural networks," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–23, 2019.
[41] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng.*, 2018, pp. 109–119.
[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[43] C. Szegedy *et al.*, "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Representations*, 2014.
[44] M. Usman, Y. Noller, C. S. Pasareanu, Y. Sun, and D. Gopinath, "NeuroSPF: A tool for the symbolic analysis of neural networks," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. Companion*, 2021, pp. 25–28.
[45] J. Wang *et al.*, "RobOT: Robustness-oriented testing for deep learning systems," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 300–311.
[46] J. Wang, J. Sun, P. Zhang, and X. Wang, "Detecting adversarial samples for deep neural networks through mutation testing," 2018, *arXiv:1805.05010*.
[47] Y. Wang, X. Ma, J. Bailey, J. Yi, B. Zhou, and Q. Gu, "On the convergence and robustness of adversarial training," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, pp. 6586–6595, 2019.
[48] J. Wei and K. Zou, "EDA: Easy data augmentation techniques for boosting performance on text classification tasks," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 6382–6388.
[49] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *Proc. 24th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2018, pp. 408–426.
[50] Z. Wu *et al.*, "MoleculeNet: A benchmark for molecular machine learning," *Chem. Sci.*, vol. 9, pp. 513–530, 2018.
[51] S. Yan *et al.*, "Correlations between deep neural network model coverage criteria and model quality," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 775–787.
[52] Y. Yu, Y.-F. Li, and Z.-H. Zhou, "Diversity regularized machine," in *Proc. 22nd Int. Joint Conf. Artif. Intel.*, 2011, pp. 1603–1608.
[53] H. Zhang and W. K. Chan, "Apricot: A weight-adaptation approach to fixing deep learning models," in *Proc. 34th Int. Conf. Autom. Softw. Eng.*, 2019, pp. 376–387.
[54] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic autonomous driving system testing," in *Proc. 33rd IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2018, pp. 132–142.
[55] X. Zhao, K. Salako, L. Strigini, V. Robu, and D. Flynn, "Assessing safety-critical systems from operational testing: A study on autonomous vehicles," *Inf. Softw. Technol.*, vol. 128, 2020, Art. no. 106393.