DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems

Xiaoning Du
Nanyang Technological University
Singapore

Xiaofei Xie*
Nanyang Technological University
Singapore

Yi Li
Nanyang Technological University
Singapore

Lei Ma*
Kyushu University
Japan

Yang Liu
Nanyang Technological University
Singapore
Zhejiang Sci-Tech University, China

Jianjun Zhao
Kyushu University
Japan

ABSTRACT

Deep Learning (DL) has achieved tremendous success in many cutting-edge applications. However, the state-of-the-art DL systems still suffer from quality issues. While some recent progress has been made on the analysis of feed-forward DL systems, little study has been done on the Recurrent Neural Network (RNN)-based stateful DL systems, which are widely used in audio, natural languages and video processing, *etc*. In this paper, we initiate the very first step towards the quantitative analysis of RNN-based DL systems. We model RNN as an abstract state transition system to characterize its internal behaviors. Based on the abstract model, we design two trace similarity metrics and five coverage criteria which enable the quantitative analysis of RNNs. We further propose two algorithms powered by the quantitative measures for adversarial sample detection and coverage-guided test generation. We evaluate *DeepStellar* on four RNN-based systems covering image classification and automated speech recognition. The results demonstrate that the abstract model is useful in capturing the internal behaviors of RNNs, and confirm that (1) the similarity metrics could effectively capture the differences between samples even with very small perturbations (achieving 97% accuracy for detecting adversarial samples) and (2) the coverage criteria are useful in revealing erroneous behaviors (generating three times more adversarial samples than random testing and hundreds times more than the unrolling approach).

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Software and its engineering** → *Software testing and debugging*.

*Xiaofei Xie (xfxie@ntu.edu.sg) and Lei Ma (malei@ait.kyushu-u.ac.jp) are the corresponding authors.

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3338954

KEYWORDS

Deep learning, recurrent neural network, model-based analysis, adversarial sample, testing

ACM Reference Format:
Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. *DeepStellar*: Model-Based Quantitative Analysis of Stateful Deep Learning Systems. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3338906.3338954

1 INTRODUCTION

Deep Learning (DL) has experienced significant progress over the past decade in many real-world applications such as image processing [12], speech recognition [20], natural language processing [43], and autonomous driving [25]. However, the state-of-the-art DL systems still suffer from quality, reliability and security problems, which could lead to accidents and catastrophic events especially when deployed on safety- and security-critical systems. We have witnessed many quality and security issues, such as one pixel attack [47], Alexa/Siri manipulation with hidden voice command [48], and the Google/Uber self-driving car accidents [18, 51]. An early-stage assessment of DL systems is of great importance in discovering defects and improving the overall product quality.

Although analysis processes and techniques are well-established for traditional software, existing techniques and toolchains could not be directly applied to DL systems, due to the fundamental differences in the programming paradigms, development methodologies, as well as the decision logic representations of the software artifacts (*e.g.*, architectures) [32, 41, 50]. To bridge the gap, research on testing [27, 32, 41, 46, 50, 56], verification [54], and adversarial sample detection [15, 19, 52] of Feed-forward Neural Networks (FNN), *e.g.*, Convolution Neural Networks (CNN) and fully connected neural networks, started to emerge recently.

Yet, the existing techniques are not specially designed to be applicable to RNN. Particularly, in contrast to FNN, RNN captures the temporal behaviors by loops and memorization with internal states to take into account the influence of previous (or future) observations. The architecture of a simple RNN is shown in Fig. 1. A simple RNN is a network of neuron-like nodes organized into successive *iterations*. It takes as inputs both the data stream and the internal state vector maintained. Instead of taking the input data as a whole, RNN processes a small chunk of data as it arrives,

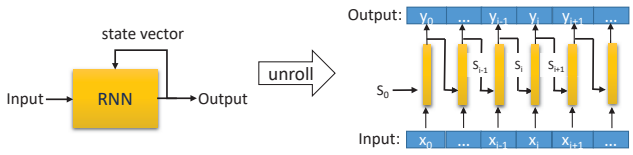


Figure 1: Architecture of a simple RNN.

and sequentially produces outputs in each iteration while updating the internal states. In other words, information in an RNN not only flows from front neural layers to the rear ones, but also from the current iteration to the subsequent ones. The stateful nature of an RNN contributes to its huge success in handling sequential data, such as audios and natural languages. At present, Long Short-Term Memory (LSTM) [22] and Gated Recurrent Unit (GRU) [10] are among the state-of-the-art and most widely used RNNs, designed with enhancement to overcome the "vanishing gradient problem" [21] that exists in the training process of most DL systems, and is aggravated by the iterative design of vanilla RNNs.

Although recent work mentions the possibility to analyze an RNN through direct unrolling, treating it as an FNN [27, 32, 50], such a strategy is still far from sufficient to handle inputs of varied lengths. Different from an FNN, where each layer has a fixed role in feature extraction, a layer in an unrolled RNN often does not preserve the same feature latent space function (or semantics) for different input sequences. Therefore, the same unrolling which works well for one input may not fit for another. In addition, there could be scalability issues when the input sequences are extremely long.

To better characterize the internal behaviors of RNNs, we propose *DeepStellar*, a general-purpose quantitative analysis framework for RNN-based DL systems. Considering its stateful nature, we first model an RNN as Discrete-Time Markov Chain (DTMC) to capture its statistical behaviors. Based on the DTMC model, we design two trace similarity metrics to quantify the prediction proximity of different inputs, and five coverage criteria to measure the adequacy of test data from different perspectives. To further demonstrate the usefulness of *DeepStellar*, we develop algorithms for two applications based the quantitative analysis, namely, RNN testing and adversarial sample detection, towards addressing highly concerned issues at present in both academia and industry.

We implemented *DeepStellar* and empirically evaluated the usefulness of the abstract model, similarity metrics and coverage criteria on four RNN-based systems from image classification to Automated Speech Recognition (ASR). Specifically, we first performed controlled experiments to evaluate the capability of the abstract model. The results demonstrate that, (1) the trace similarity metrics serve as good indicators of the discriminatory power of RNNs, and the abstract model is sensitive enough to distinguish inputs generated with very small perturbations; and (2) the coverage criteria derived from the abstract model are able to measure test adequacy and are effective in manifesting erroneous behaviors. Further, we applied the metrics and criteria on two applications, *i.e.*, adversarial sample detection and coverage-guided testing of RNNs. The results show that *DeepStellar* (1) detects 89% and 97% of the adversarial samples, respectively, for ASR and image classification systems, and (2) generates tests with high coverage and yields at most hundreds of times more adversarial samples than random testing and existing neuron coverage guided testing with unrolling [50].

The main contributions of this paper are summarized as follows:

- We propose to formalize an RNN-based stateful DL system as a DTMC model, to characterize the internal states and dynamic behaviors of the systems.
- Based on the DTMC abstraction, we design two similarity metrics and five coverage criteria for stateful DL systems, which are among the first to quantify sample differences and test data adequacy for RNNs.
- With *DeepStellar*, we design two algorithms for detecting adversarial samples and conducting guided testing for RNNs based on the metrics and criteria.
- We conduct in-depth evaluation to demonstrate the usefulness of *DeepStellar* with controlled experiments as well as two typical real-world applications.

2 OVERVIEW

Fig. 2 summarizes the workflow of our approach, including the *abstract model construction* of RNN, different *quantitative measures* defined over the abstract model, and two *applications* to detect and generate adversarial samples of RNNs.

The abstract model construction module takes a trained RNN as input and analyses its internal behaviors through profiling. The inputs for profiling are from the training data, which can best reflect the characteristics of a trained RNN model. Specifically, each input sequence is profiled to derive a *trace*, *i.e.*, a sequence of RNN state vectors. After the profiling, we can get a set of traces which record the states visited and transitions taken during the training stage.

In practice, the internal state space of an RNN and the number of traces enabled by the training data are often beyond our analysis capability. Therefore, we perform abstraction over the states and traces to obtain an abstract model that captures the global characteristics of the trained network. At the state level, we apply Principle Component Analysis (PCA) [26] to reduce the dimensions of the state vectors and keeps the first k most dominant components. For each of the k dimensions, we further partition it into m equal intervals. At the transition level, we consolidate concrete transitions into abstract ones according to the abstract states. We also take into account the frequencies of different transitions at each state and effectively derive a Discrete-Time Markov Chain (DTMC) [38] model for the trained RNN.

Based on the abstract model, we design two metrics for evaluating the trace similarity induced by different inputs, and five coverage criteria to facilitate the systematic testing of RNNs. The metrics and coverage criteria are designed from both the state- and transition-level. Specifically, the trace similarity metrics include *state-based trace similarity* (SBTSim) and *transition-based trace similarity* (TBTSim). The coverage criteria include the *basic state coverage* (BSCov), *n-step state boundary coverage* (*n*-SBCov), *weighted state coverage* (WSCov), *basic transition coverage* (BTCov), and *weighted transition coverage* (WTCov).

We then apply the metrics and criteria on two applications, *i.e.*, the *adversarial sample detection* and *coverage-guided testing*, both of which aim to mitigate the threats from adversarial samples. With the similarity metrics, we propose an approach to detect adversarial

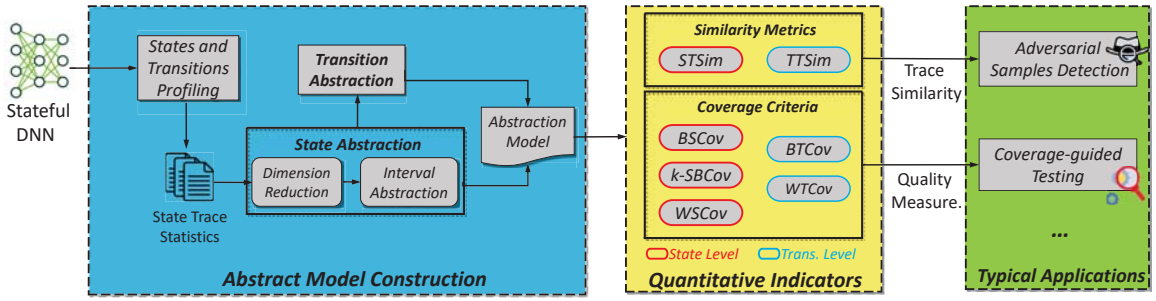


Figure 2: Overview of *DeepStellar* and its typical applications.

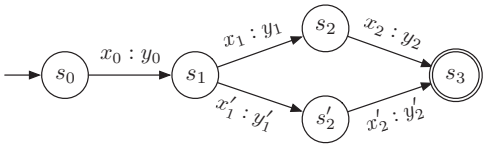


Figure 3: An example FST representing two traces.

samples at runtime. With the coverage criteria, we develop a testing framework to guide test generation with the aim to improve coverage and uncover defects for quality assurance. The two techniques are complementary to each other. The testing technique aims to generate unseen adversarial samples that help developers analyze and improve the robustness of the model. The adversarial sample detection technique is able to identify malicious inputs and prevent potential damages at runtime.

3 STATE TRANSITION MODELING OF RNN

3.1 RNN Internal States and State Transitions

Following [42], we represent a neural network abstractly as a differentiable parameterized function $f(\cdot)$. The input to an RNN is a sequence $\mathbf{x} \in \mathcal{X}^N$, where $\mathcal{X}$ is the input domain and N is the length of the sequence. Let $x_i \in \mathcal{X}$ be the i-th element of the sequence $\mathbf{x}$. Then, when passing $\mathbf{x}$ into an RNN, it maintains a state vector $\mathbf{s} \in \mathcal{S}^N$ with $s_0 = \mathbf{0}$ and $(s_{i+1}, y_i) = f(s_i, x_i)$, where $\mathcal{S}$ is the domain of the hidden state, $s_i \in \mathcal{S}$ is the hidden state of RNN at the i-th iteration, and $y_i \in \mathcal{O}$ is the corresponding output at that step. We use s_i^d to denote the d-th dimension of the state vector s_i.

Naturally, each input sequence $\mathbf{x}$ induces a finite sequence of state transitions $\mathbf{t}$, which we define as a *trace*. The i-th element in a trace $\mathbf{t}$, denoted by t_i, is the transition from s_i to s_{i+1} after accepting an input x_i and producing an output y_i. A Finite State Transducer (FST) [16] can be used to represent a collection of traces more compactly [23] as defined below.

Definition 1. *An FST is a tuple $(\mathcal{S}, \mathcal{X}, \mathcal{O}, I, F, \delta)$ such that $\mathcal{S}$ is a non-empty finite set of states, $\mathcal{X}$ is the input alphabet, $\mathcal{O}$ is the output alphabet, $I \subseteq \mathcal{S}$ is the set of initial states, $F \subseteq \mathcal{S}$ is the set of final states, and $\delta \subseteq \mathcal{S} \times \mathcal{X} \times \mathcal{O} \times \mathcal{S}$ is the transition relation.*

For example, Fig. 3 shows a simple FST representing two traces, namely, $s_0 s_1 s_2 s_3$ and $s_0 s_1 s'_2 s_3$ with s_0 being the initial state and s_3 being the final state. The first trace takes an input sequence $x_0 x_1 x_2$ and emits an output sequence $y_0 y_1 y_2$; the second trace takes an input sequence $x_0 x'_1 x_2$ and emits an output sequence $y_0 y'_1 y_2$.

3.2 Abstract State Transition Model

The number of states and traces enabled while training an RNN can be huge. To effectively capture the behaviors triggered by a large number of input sequences and better capture the global characteristics of the trained network, we introduce an *abstract state transition model* in this paper. The abstract model over-approximates the observed traces induced of an RNN and has a much smaller set of states and transitions compared with the original one. The abstraction is also configurable – one can trade-off between the size and precision of the model so that the abstract model is still able to maintain useful information of the input sequences for particular analysis tasks. To obtain an abstract model for a trained RNN, we abstract over both the states and the transitions.

State Abstraction. Each *concrete state* s_i is represented as a vector $(s_i^1, \ldots, s_i^m)$, usually in high dimension (*i.e.*, m could be a large number). Intuitively, an *abstract state* represents a set of concrete states which are close in space. To obtain such a state abstraction, we first apply the Principle Component Analysis (PCA) [26] to perform an orthogonal transformation on the concrete states – finding the first k principle components (*i.e.*, axes) which best distinguish the given state vectors and ignore their differences on the other components. This is effectively to project all concrete states onto the chosen k-dimensional component basis (denoted as PCA-k).

Then, we split the new k-dimensional space into m^k *regular grids* [49] such that there are m equal-length intervals on each axis:

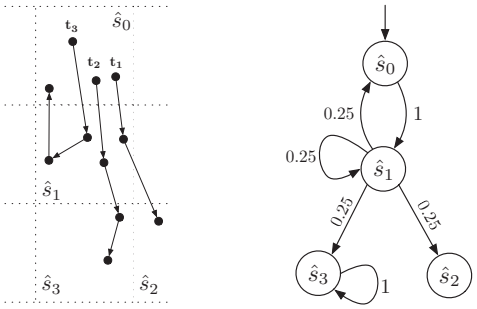
$$e_i^d = [lb_d + i \times \frac{ub_d - lb_d}{m}, lb_d + (i+1) \times \frac{ub_d - lb_d}{m}],$$

where e_i^d represents the i-th interval on the d-th dimension, lb_d and ub_d are the lower and upper bounds of all state vectors on the d-th dimension, respectively. In this way, all concrete states s_i which fall within the same grid are mapped to the same abstract state: $\hat{s} = \{s_i | s_i^1 \in e^1 \wedge \cdots \wedge s_i^k \in e^k\}$. We denote the set of all abstract states as $\hat{\mathcal{S}}$. Noticeably, the precision of the state abstraction can easily be configured by tuning the parameters k and m.

Let $j = I^d(\hat{s})$ be the index of $\hat{s}$ on the d-th dimension such that for all $s \in \hat{s}$, s^d falls in e_j^d $(0 \le j < m)$. For any two abstract states $\hat{s}$ and $\hat{s}'$, we define their *distance* as:

$$Dist(\hat{s}, \hat{s}') = \Sigma_{d=1}^{k} |I^d(\hat{s}) - I^d(\hat{s}')|.$$

This definition can also be generalized to include space beyond the lower and upper bounds.



(a) Example concrete traces. (b) DTMC abstraction.

Figure 4: A set of concrete traces and their corresponding abstract state transition model.

Transition Abstraction. Once the state abstraction is computed, a concrete transition between two concrete states can be mapped as a part of an *abstract transition*. An abstract transition represents a set of concrete transitions which share the same source and destination abstract states. In other words, there is an abstract transition between two abstract states $\hat{s}$ and $\hat{s}'$ if and only if there exists a concrete transition between s and s' such that $s \in \hat{s} \wedge s' \in \hat{s}'$. The set of all abstract transitions is denoted as $\hat{\delta} \subseteq \hat{\mathcal{S}} \times \hat{\mathcal{S}}$.

For instance, Fig. 4a depicts three concrete traces, *i.e.*, $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_3$, where states are shown as dots and transitions are directed edges. The grids drawn in dashed lines represent the abstract states, *i.e.*, $\hat{s}_0$, $\hat{s}_1$, $\hat{s}_2$, and $\hat{s}_3$, each of which is mapped to a set of concrete states inside the corresponding grid. The set of abstract transitions is, therefore, $\{(\hat{s}_0, \hat{s}_1), (\hat{s}_1, \hat{s}_0), (\hat{s}_1, \hat{s}_1), (\hat{s}_1, \hat{s}_2), (\hat{s}_1, \hat{s}_3), (\hat{s}_3, \hat{s}_3)\}$.

3.3 Trace Similarity Metrics

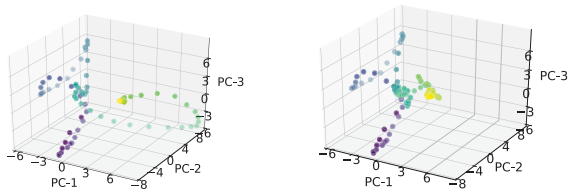
To precisely compare two input sequences, we define the *trace similarity metrics* to quantify the proximity of their induced state transitions on the abstract model. Given an abstract model M and an input $\mathbf{x}$, we denote the set of abstract states and transitions covered by $\mathbf{x}$ as $\hat{\mathcal{S}}_\mathbf{x}$ and $\hat{\delta}_\mathbf{x}$. Then, the *state- and transition-based trace similarity metrics* for the two inputs $\mathbf{x}$ and $\mathbf{y}$ are defined based on the Jaccard indices of their states and transitions covered, respectively:

$$\text{STSIM}_\text{M}(\mathbf{x}, \mathbf{y}) = \frac{|\hat{\mathcal{S}}_\mathbf{x} \cap \hat{\mathcal{S}}_\mathbf{y}|}{|\hat{\mathcal{S}}_\mathbf{x} \cup \hat{\mathcal{S}}_\mathbf{y}|}, \quad \text{TTSIM}_\text{M}(\mathbf{x}, \mathbf{y}) = \frac{|\hat{\delta}_\mathbf{x} \cap \hat{\delta}_\mathbf{y}|}{|\hat{\delta}_\mathbf{x} \cup \hat{\delta}_\mathbf{y}|}.$$

The trace similarity metrics range over $[0, 1]$, where 0 indicates disjoint sets (*i.e.*, traces induced by $\mathbf{x}$ and $\mathbf{y}$ are totally different), while 1 indicating equal sets (*i.e.*, the traces are similar).

Fig. 5 shows the concrete traces on an RNN-based ASR model induced by two speech input samples, Fig. 5a ("this book is about science") and Fig. 5b ("this book is about literature"). The darker dots appear earlier in the sequence, and vice versa. We can see a clear difference of the two at later parts of the sequence. The two concrete traces are then projected onto the 3-dimensional space under the PCA-3 abstraction with 5 intervals on each dimension, to calculate the trace similarities. The state- and transition-based trace similarities of the two inputs are 0.71 and 0.64, respectively.

Each input sequence in the training set yields a concrete trace of the RNN model. The abstract state transition model captures all the concrete traces enabled from training data (or its representative parts) and other potential traces which have not been enabled. The



(a) "This book is about science." (b) "This book is about literature."

Figure 5: Visualization of concrete traces of two audio over an RNN-based ASR model with the PCA-3 abstraction.

defined state and transition abstraction make the resulting abstract model represent an over-approximation and generalization of the observed behaviors of the trained RNN model.

3.4 Representing Trained RNN as a Discrete-Time Markov Chain

To also take into account the likelihood of transitions at different states, we augment the abstract model with transition probabilities, effectively making it a Discrete-Time Markov Chain (DTMC).

Definition 2. *A DTMC is a tuple* $(\hat{\mathcal{S}}, I, \hat{\mathcal{T}})$, *where* $\hat{\mathcal{S}}$ *is a set of abstract states,* I *is a set of initial states, and* $\hat{\mathcal{T}} : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \mapsto [0, 1]$ *is the* transition probability function *which gives the probability of different abstract transitions.*

We write $\Pr(\hat{s}, \hat{s}')$ to denote the conditional probability of visiting $\hat{s}'$ given the current state $\hat{s}$, such that $\Sigma_{\hat{s}' \in \hat{\mathcal{S}}} \Pr(\hat{s}, \hat{s}') = 1$. We define the transition probability as the number of concrete transitions from $\hat{s}$ to $\hat{s}'$ over the number of all outgoing concrete transitions from $\hat{s}$, *i.e.*, $\Pr(\hat{s}, \hat{s}') = \frac{|\{(s,s') | s \in \hat{s} \wedge s' \in \hat{s}'\}|}{|\{(s,\_) | s \in \hat{s}\}|}$. For example, Fig. 4b shows the abstract state transition model for the concrete traces in Fig. 4a as a DTMC. The abstract transitions are labeled with their transition probabilities. For instance, since all outgoing transitions at $\hat{s}_0$ end in $\hat{s}_1$, the transition probability from $\hat{s}_0$ to $\hat{s}_1$ is 1. There are four possible outgoing transitions at $\hat{s}_1$, *i.e.*, $\{(\hat{s}_1, \hat{s}_1), (\hat{s}_1, \hat{s}_2), (\hat{s}_1, \hat{s}_3), (\hat{s}_1, \hat{s}_0)\}$. Hence, the transition probability from $\hat{s}_1$ to $\hat{s}_2$ is computed as $\Pr(\hat{s}_1, \hat{s}_2) = \frac{1}{4}$. Computation for other abstract transitions are similar.

As is shown in the example, a DTMC model is constructed by first applying the state and transition abstractions on a set of concrete traces, and then computing transition probability distributions for each abstract state. The time complexity of the abstraction step depends on the number of concrete traces, while the complexity for computing the transition probabilities only depends on the number of abstract transitions.

4 COVERAGE CRITERIA FOR RNN

Inspired by traditional software testing, we propose a set of testing coverage criteria for RNNs based on the abstract state transition model. The goal of the RNN coverage criteria is to measure the sufficiency of test data in exercising the trained as well as the unseen behaviors. The state and transition abstractions are designed to reflect the internal network configurations at a certain point as well as the temporal behaviors of the network over time, respectively. Therefore, to maximize the chance of discovering defects in stateful

neural networks, one should combine coverage criteria based on both the state and transition abstractions to systematically generate comprehensive and diverse test suites.

Let $M = (\hat{\mathcal{S}}, I, \hat{\mathcal{T}})$ be an abstract model of the trained RNN represented as a DTMC. Let $T = \{\mathbf{x}_0, \ldots, \mathbf{x}_n\}$ be a set of test input sequences. We define both the *state-level* and *transition-level* coverage of T to measure how extensively T exercises the states and transitions of M, respectively.

4.1 State-Level Coverage Criteria

The state-level coverage criteria focus on the internal states of the RNN. The set of abstract states $\hat{\mathcal{S}}$ represents a space generalization of the visited states obtained from training data (or its representative parts), which is referred to as the *major function region* [32]. The space outside the major function region is never visited by the training data, and thus represents the *corner-case region* [32]. The test data should cover the major function region extensively to validate the trained behaviors and cover the corner-case region sufficiently in order to discover defects in unseen behaviors.

Basic State Coverage. Given an RNN abstract model M and a set of test inputs T, the *basic state coverage* measures how thoroughly T covers the major function region visited while training. To quantify this, we compare the set of abstract states visited by the training inputs and the test inputs, denoted by $\hat{\mathcal{S}}_M$ and $\hat{\mathcal{S}}_T$, respectively. Then, the basic state coverage is given by the number of abstract states visited by both the training and the test inputs over the number of states visited by the training inputs,

$$\text{BSCov}(T, M) = \frac{|\hat{\mathcal{S}}_T \cap \hat{\mathcal{S}}_M|}{|\hat{\mathcal{S}}_M|}.$$

Weighted State Coverage. The basic state coverage treats every state with equal weights. During training, not all states are visited equally often and one may want to emphasize more on some states than the others. To take into account the frequencies of different states and be able to assign weights to states, we define the *weighted state coverage* and allow users to specify a *weight function*. The default weight of an abstract state $\hat{s}$ is defined as the relative frequency of it among all the abstract states, *i.e.*, $w(\hat{s}) = \frac{|\{s | s \in \hat{s}\}|}{|\mathcal{S}|}$, where $\mathcal{S}$ is the set of all distinct concrete states. Then, the weighted state coverage is defined as:

$$\text{WSCov}(T, M) = \frac{\Sigma_{\hat{s} \in \hat{\mathcal{S}}_T \cap \hat{\mathcal{S}}_M} w(\hat{s})}{\Sigma_{\hat{s} \in \hat{\mathcal{S}}_M} w(\hat{s})}.$$

In practice, the weight function can be defined differently according to specific needs. For example, when a constant function is chosen, the weighted state coverage is equivalent to the basic one. In § 6.3, we evaluate two weight functions, including the default and the reversed one which assigns larger weight to less visited states.

***n*-Step State Boundary Coverage.** The test data may also trigger new states that are never visited during training. The *n-step state boundary coverage* measures how well the corner-case regions are covered by the test inputs T. The corner-case regions $\hat{\mathcal{S}}_{M^c}$ are the set of abstract states outside of $\hat{\mathcal{S}}_M$, which have non-zero distances from any states in $\hat{\mathcal{S}}_M$. Then $\hat{\mathcal{S}}_{M^c}$ can be further divided into different boundary regions defined by their distances from $\hat{\mathcal{S}}_{M^c}$. For example, the *n-step boundary region*, $\hat{\mathcal{S}}_{M^c}(n)$, contains all abstract states which have a minimal distance n from $\hat{\mathcal{S}}_M$, or more formally, $\hat{\mathcal{S}}_{M^c}(n) = \{\hat{s} \in \hat{\mathcal{S}}_{M^c} \,|\, \min_{\hat{s}' \in \hat{\mathcal{S}}_M} Dist(\hat{s}, \hat{s}') = n\}$.

The n-step state boundary coverage is defined as the ratio of states visited by the test inputs in the boundary regions of at most n steps away from $\hat{\mathcal{S}}_M$:

$$n\text{-SBCov}(T, M) = \frac{|\hat{\mathcal{S}}_T \cap \bigcup_{i=1}^{n} \hat{\mathcal{S}}_{M^c}(i)|}{|\bigcup_{i=1}^{n} \hat{\mathcal{S}}_{M^c}(i)|}.$$

4.2 Transition-Level Coverage Criteria

The state-level coverage indicates how thorough the internal states of an RNN are exercised but it does not reflect the different ways transitions have happened among states in successive time steps. The transition-level coverage criteria targets at the abstract transitions activated by various input sequences and a higher transition coverage shows that the inputs are more adequate in triggering diverse temporal dynamic behaviors.

Basic Transition Coverage. To quantify transition coverage, we compare the abstract transitions exercised during both the training and testing stages, written as $\hat{\delta}_M$ and $\hat{\delta}_T$, respectively. The *basic transition coverage* is defined as:

$$\text{BTCov}(T, M) = \frac{|\hat{\delta}_T \cap \hat{\delta}_M|}{|\hat{\delta}_M|}.$$

Basic transition coverage subsumes basic state coverage. In other words, for any abstract model M, a test input T satisfies basic transition coverage regarding to M, also satisfies basic state coverage.

Weighted Transition Coverage. Similar as the state-level coverage, we can calculate the *weighted transition coverage* by considering the relative frequency of each transition. More formally,

$$\text{WTCov}(T, M) = \frac{\Sigma_{(\hat{s}, \hat{s}') \in \hat{\delta}_T \cap \hat{\delta}_M} w(\hat{s}, \hat{s}')}{\Sigma_{(\hat{s}, \hat{s}') \in \hat{\delta}_M} w(\hat{s}, \hat{s}')}.$$

where the weight function can be configured similarly as in the weighted state coverage. By default, the weight of a transition $(\hat{s}, \hat{s}')$ is computed as transition probability defined in § 3.4.

5 APPLICATIONS

To demonstrate the usefulness of the abstract model and the proposed quality measures, we apply them in finding adversarial samples for RNNs in two scenarios: (1) adversarial sample detection to identify adversarial inputs at runtime, and (2) coverage-guided testing to generate unseen adversarial samples offline.

5.1 Adversarial Sample Detection for RNNs

Adversarial sample detection aims to check whether a given input is an adversarial sample at runtime. We propose to use the trace similarity metrics to measure the behavioral differences between two inputs. Based on this idea, we develop a new approach to detect adversarial samples for RNNs.

Given a target sample i, we define a *reference sample* r such that the RNN gives the same predictions for both i and r. The traces derived from the original and the reference samples are typically similar when i is benign. However, when i is adversarial, the trace difference between the two samples can be much larger. With such

Algorithm 1: Training an Adversarial Detection Classifier

```
   input  : D: RNN-based DL system, M: Abstract model of D
   output : C: A classifier for detecting adversarial samples
1  Prepare benign set B, adversarial set A and reference set R;
2  dis_b ← ∅;
3  for b ∈ B do
4      R' ← select(R, b) ;
5      vec ← ∅;
6      for r ∈ R' do
7          (r_1, state_vec1) ← predict(R, r);
8          (r_2, state_vec2) ← predict(R, b);
9          j ← TraceSimilarity(state_vec1, state_vec2, M);
10         vec ← vec ⋃ {j};
11     d ← average(vec);
12     dis_b ← dis_b ⋃ {d};
13 Compute dis_a similar with dis_b;
14 C ← LinearRegressionClassifer(dis_a, dis_b)
```

Algorithm 2: Coverage guided testing of RNN

```
   input  : I: Initial seeds, D: RNN-based DL system, M: Abstract
            model of D
   output : F: Failed tests, Q: Test queue
1  F ← ∅;
2  Q ← I;
3  while a ← Select(Q) do
4      Randomly pick transformation t with a random parameter p ;
5      A = mutate(t, p, a);
6      for a' ∈ A do
7          (result, state_vec) ← predict(R, a');
8          cov ← CovAnalysis(state_vec, M);
9          if Failed(a', result) then
10             F ← F ⋃ {a'}
11         else if CoverageIncrease(cov, Q) then
12             Q ← Q ⋃ a';
13             UpdateCoverage(Q);
```

insight, we propose to detect adversarial sample based on the trace difference from its reference sample.

We use a learning-based approach (Algorithm 1) to train a classifier. The inputs include an RNN-based system D and an abstract model M. We first collect a set of benign samples B and a set of adversarial samples A. The set of reference samples R are also prepared for comparison (more details in the next paragraph), and the trace similarities between benign/adversarial samples and the reference samples are calculated. For each benign sample b, the *select* function obtains a group of corresponding reference samples R' from the reference samples R. Based on this, we compute the trace similarity between b and each reference sample $r \in R'$ (Lines 6- 10) and take the average similarity (Line 11) to represents the distance between b and the set of reference samples R'. The distance is added into the benign distance list dis_b (Line 12). Similarly, we compute the adversarial distance list dis_a (Line 13). With dis_a and dis_b, a linear regression classifier C is learned. Given a new input i, we compute the similarity d_i between i and the reference samples, and rely on $C(d_i)$ to indicate whether i is a benign or adversarial.

We apply and evaluate the detection algorithms on two domains, namely, ASR and image classification. The approaches used to generate reference samples are as follows: for ASR, given an audio input a that is transcribed to texts t by the RNN, we generate the reference audios using off-the-shelf text-to-speech engines (*e.g.*, Google cloud text-to-speech [4]) that generate the audio t with correct and clear pronouncing. For image classification, given a new image i with prediction result c (*i.e.*, the image i belongs to class c), the reference images are selected from the training data such that they share the same label as c.

5.2 Coverage-Guided Testing of RNNs

In this section, we propose a Coverage-Guided Testing (CGT) technique which aims to generate adversarial samples incorrectly recognized by an RNN. CGT uses the proposed coverage criteria (§ 4) to guide the test generation and evaluates the quality of the tests from different perspectives. During the testing process, CGT maintains a test queue. In each run, it selects a seed (*i.e.*, test case) from the queue and generates multiple mutants. A mutant is an adversarial sample if it is predicted incorrectly by the network. Otherwise, if the mutant improves the coverage, it is then retained as an interesting seed and added back to the queue.

Algorithm 2 presents the process to generate tests for RNNs. The inputs include the initial seeds I, the RNN-based DL system D and the abstract model M. The outputs are benign tests and failed tests for which D gives correct and incorrect inference respectively. The initial test queue contains a set of initial seeds. In each run, CGT selects one input a from the test queue (Line 2), and randomly picks a transformation function t with parameter p (Line 4). Then, a set of new samples A are generated under transformation t (Line 5). For each new sample a', CGT first obtains the concrete trace by letting D do the inference (Line 7) and then calculates the coverage information over the abstract model (Line 8). If the inference is incorrect, a' is added into the failed test set F (Line 10). If a' is correctly predicted and covers new states or transitions, CGT puts it in the test queue and updates the coverage criteria of all tests currently in the queue (Line 12- 13).

A challenge in DL testing is the lack of oracle that tells the ground-truth label of any new mutant. Mutation operators are often specific to the application domains. For CGT, we mainly focus on image classification and apply the metamorphic mutation strategy [55] for generating new mutants that would keep the prediction meaning from the human's perspective during testing.

6 EVALUATION

To demonstrate the usefulness of the proposed techniques, we implemented *DeepStellar* in Pythonbased on the Keras (2.2.4) [11] with TensorFlow (1.4, 1.8 and 1.11) [7]. We first study whether the abstract model is able to characterize the stateful behaviors of RNNs (*i.e.*, **RQ1** & **RQ2**). Based on this, we further evaluate the usefulness of the proposed quantitative measures on the two applications (*i.e.*, **RQ3** & **RQ4**). Specifically, we leverage *DeepStellar* to investigate the following research questions:

RQ1: Are the proposed trace similarity metrics suitable indicators for the discriminatory power of RNNs (*i.e.*, sensitive to even small perturbations on inputs)?

Table 1: Subject model information.

Subject Model	Kernel RNN		# Trainable	Acc. (%)	
	Type	State vec. shape	Parameters	Train.	Test.
DeepSpeech 0.1.1	Bi-LSTM	(None, 4096)	122,740,765	-	-
DeepSpeech 0.3.0	LSTM	(None, 2048)	47,228,957	-	11.00
MNIST-LSTM	LSTM	(None, 128)	81,674	99.69	98.66
MNIST-GRU	GRU	(None, 128)	61,578	99.70	98.61

RQ2: How sensitive are different coverage criteria for capturing erroneous behaviors of RNNs?
RQ3: How useful is the trace similarity-based detection algorithm for detecting adversarial samples of RNNs?
RQ4: How effective is the coverage-guided testing in achieving high coverage and generating adversarial samples of RNNs?

6.1 Experiment Settings

Models and Dataset. We selected four RNN-based DL models including two ASR models and two image classification models, which cover popular RNN variants (see Table 1). The trainable parameter size often reflects the complexity of models, and the models we select range from small-scale ones with about 60k parameters, to practical-sized ones with over 100 million parameters. The highest dimension of the RNN state vectors (Column "State vec. shape") hits 4,096, indicating the high complexity of the models.

For the ASR tasks, we selected two versions of Mozilla pre-trained DeepSpeech [2] (*i.e.*, `0.1.1` and `0.3.0`) that are among the state-of-the-art open source ASR models with different types of RNN core. `DeepSpeech-0.1.1` adopts a bi-directional LSTM, and `DeepSpeech-0.3.0` uses a one-directional LSTM. The state vectors of both models are with a high dimension of 64-bit floating point type. For the image classification task, we followed the instructions and trained two RNN-based classifiers on MNIST dataset (*i.e.*, `MNIST-LSTM`, `MINIST-GRU`) that achieve competitive accuracy.These two models are relatively lightweight, whose internal states are conveyed via 128-dimensional vectors of 32-bit floating points.

Abstract Model Construction. For ASR models, we use *Common Voice* [1] training dataset to perform the profiling, which is used for the training of the DeepSpeech models. Overall, there are 193,284 audios in the dataset. Each sample is processed by both models to collect the state traces. As for the image classification models, we use the official MNIST training dataset that contains 60,000 images.

For the PCA transformation, due to the huge size of state vectors for ASR models, we randomly selected 20% of state vectors to fit the PCA model, and use it for all further analysis. For image models, we use all of the state vectors. The model abstraction parameters k and m can be configured to generate DTMC models with different granularity. Table 2 summarizes 13 different configurations we evaluated, as well as the number of abstract states and transitions in each obtained DTMC model. Note that we use (k, m) to represent a configuration with k dimensions and m partitions.

Data Preparation. For a comprehensive evaluation of the proposed metrics and coverage criteria, we prepared three types of samples: 1) original benign samples from the test data, 2) perturbed samples which are generated by a slight perturbation on the original benign samples, and 3) adversarial samples from the original benign samples. For ASR models, we use *word error rate* (WER) to measure the inference precision. In particular, benign samples are with WER of zero. The perturbed samples would have a relatively smaller WER while the targeted adversarial samples have a larger WER. For image classification tasks, the perturbed samples we generate are with slight perturbations but still remain benign.

Table 2: Abstract model details under different configurations for each studied RNN.

Config.	DeepSpeech-0.1.1		DeepSpeech-0.3.0		MNIST-LSTM		MNIST-GRU	
(k, m)	# St.	# Trans.	# St.	# Trans.	# St.	# Trans.	# St.	# Trans.
(2, 5)	26	187	27	159	28	162	28	192
(2, 10)	88	1,067	86	755	93	745	92	1,203
(2, 20)	325	8,185	308	4,958	340	4,531	334	8,740
(2, 40)	1,221	74,936	1,154	38,653	1,267	27,657	1,241	57,506
(2, 80)	4,700	733,007	4,397	327,480	4,719	105,529	4,656	167,640
(3, 5)	107	1,416	96	1,269	109	938	109	1,188
(3, 10)	595	13,643	533	10,836	632	6,186	606	9,973
(3, 20)	3,811	179,925	3,291	128,144	3,806	39,148	3,713	72,989
(3, 40)	26,154	2,480,927	21,970	1,708,455	21,458	128,225	22,829	183,478
(3, 80)	183,724	18,408,782	145,602	15,232,230	78,259	200,567	101,417	218,765
(6, 10)	65,550	2,567,944	55,055	2,040,096	24,276	96,749	34,423	139,595
(6, 20)	1,373,236	23,057,145	1,108,832	21,164,285	110,325	185,223	149,801	209,887
(6, 40)	15,804,002	38,324,511	14,258,035	37,594,508	194,765	213,978	215,680	223,505

Initially, we randomly selected 100 benign audios and 100 benign images separately from their test datasets, from which we generate the perturbed and adversarial samples. For each ASR model, we generate 10,000 perturbed audios from original benign ones with existing audio data augmentation techniques [5] (*i.e.*, speed and volume adjustment, low/high-frequency filtering, noise blending). Finally, we only successfully generate adversarial samples for `DeepSpeech-0.1.1` because there exists a compatibility issue between `DeepSpeech-0.3.0` and the adversarial attack tools [8] we used. It is worth noting that the generation of targeted adversarial audios is rather computationally intensive and time-consuming. To be specific, we select the 11 commands [13] as the targets and generate 1,100 (100 seeds× 11 targets) adversarial audios, which took about 12 days in total on 4 GPUs (*i.e.*, 48 days V100 GPU time). For each MNIST model, we also generate 10,000 benign perturbed samples with existing image transformation techniques [50] (*i.e.*, image contrast, brightness, translation, scaling, shearing, rotation and add white noise). Besides, we generate 10,000 adversarial samples with each state-of-the-art attack tool, including, *FGSM* [17], *BIM* [28] and *DeepFool* [37].

For both audio and image case, we set conservative parameters for transformation so that the perturbation on original samples is slight and imperceptible. Note that all the adversarial samples are also with minimal perturbations and not perceptible by human.

Coverage Criteria Instances. For the n-SBCov criteria, we empirically study two instances with $n = 3$ and $n = 6$, denoted as 3-SBCov and 6-SBCov, respectively. For the WSCov and WTCov, besides the default weight function, which assigns larger weights to states or transitions with high visiting frequency during profiling, we introduce another weight function to assign smaller weights to more frequently visited states and transitions by inverting the original weights. We use WSCov and WTCov to denote the criteria with default weight function, and refer WSCov' and WTCov' to the inverted ones. The criteria allow observing how the test data cover states and transitions that have high/low visiting frequency.

All the experiments were run on a server with the Ubuntu 16.04 system with 28-core 2.0GHz Xeon CPU, 196 GB RAM and 4 NVIDIA Tesla V100 16G GPUs.

Table 3: Correlation of trace similarities and prediction diff.

Config.	DeepSpeech-0.1.1		DeepSpeech-0.3.0		MNIST-LSTM		MNIST-GRU	
(k, m)	$\rho(st.)$	$\rho(tr.)$	$\rho(st.)$	$\rho(tr.)$	$U(st.)$	$U(tr.)$	$U(st.)$	$U(tr.)$
(2, 5)	-0.25	-0.28	-0.36	-0.44	2.99.E+08	**2.79.E+08**	2.70.E+08	**2.62.E+08**
(2, 10)	-0.27	-0.35	-0.44	-0.50	3.03.E+08	2.76.E+08	2.53.E+08	2.48.E+08
(2, 20)	-0.35	-0.38	-0.46	-0.49	2.84.E+08	2.55.E+08	2.49.E+08	2.33.E+08
(2, 40)	-0.37	-0.26	-0.43	-0.47	2.50.E+08	2.30.E+08	2.29.E+08	2.05.E+08
(2, 80)	-0.30	-0.16	-0.40	-0.36	2.30.E+08	2.20.E+08	2.04.E+08	1.92.E+08
(3, 5)	-0.34	-0.38	-0.47	-0.52	**3.11.E+08**	**2.92.E+08**	**2.89.E+08**	**2.74.E+08**
(3, 10)	-0.39	**-0.39**	-0.52	-0.50	**3.04.E+08**	2.64.E+08	**2.73.E+08**	2.45.E+08
(3, 20)	-0.38	-0.34	-0.51	**-0.53**	2.74.E+08	2.35.E+08	2.44.E+08	2.09.E+08
(3, 40)	-0.31	-0.15	-0.51	-0.43	2.35.E+08	2.14.E+08	2.05.E+08	1.82.E+08
(3, 80)	-0.23	-0.09	-0.41	-0.25	2.17.E+08	2.11.E+08	1.93.E+08	1.89.E+08
(6, 10)	**-0.54**	**-0.47**	**-0.62**	**-0.59**	2.65.E+08	2.18.E+08	2.23.E+08	1.95.E+08
(6, 20)	**-0.40**	-0.24	**-0.59**	-0.43	2.25.E+08	2.14.E+08	1.89.E+08	1.90.E+08
(6, 40)	-0.06	-0.06	-0.40	-0.13	2.14.E+08	2.09.E+08	1.89.E+08	1.89.E+08

6.2 RQ1: Trace Similarity

Setup. We perform a statistical analysis on the correlation between the trace similarity and the prediction difference over the slightly perturbed samples and their original benign samples. The difference is difficult to capture because the sample and its slightly perturbed counterpart are perceived almost the same from human perceptions. We compute the prediction difference of ASR with the word-level Levenshtein distance [6] of their transcripts, and the prediction difference in image classification by checking whether they belong to different classes.

For audios, we use the 10,000 perturbed samples, which are with various Levenshtein distances compared with their original seeds. For the image case, we take the 10,000 perturbed samples and also randomly select another 10,000 samples from all the generated adversarial samples. This is to include both correctly and wrongly inferring perturbed images. For the statistical analysis, we use Spearman rank-order correlation [44] (denoted as ρ), to analyze the monotonic association between two variables, for the ASR models; and we use Mann-Whitney U test [35] (denoted as U), to check the binary association, for MNIST models.

Results. Table 3 shows the results of the correlation between trace similarity and prediction difference measured over the perturbed data, where Column $\rho(st.)$ and Column $U(st.)$ represent the results of StSim; and Column $\rho(tr.)$ and Column $U(tr.)$ represent the results of TTSim. The best two results of each column are highlighted in bold font. All reported correlations are statistically significant (with $p << 0.01$). Negative association of Spearman correlation indicates that the larger the similarity metrics, the less different the predicted transcripts would be. For MNIST models, the Mann-Whitney U test results indicate that when measuring the trace similarity compared with the original benign samples, perturbed samples obtain significantly larger values than adversarial ones.

Answer to RQ1: Both state- and transition-level trace similarity metrics are capable of capturing the prediction difference even for slightly perturbed samples. Thus, trace similarity could be useful for detecting adversarial samples (See RQ4).

6.3 RQ2. Coverage Criteria

Setup. In this experiment, we evaluate the sensitivity of the proposed coverage criteria to adversarial samples. The abstraction configurations used in RQ2 are selected based on the RQ1 results. In order to better differentiate various coverage criteria, three configurations were selected from RQ1, which are most sensitive to minor perturbations according to either StSim or TTSim. We compare the coverage results of the 100 original benign samples (denote as *O*), and the coverage achieved by including perturbed samples (denoted as *O+P*) or adversarial samples (denoted as *O+A*) (see Column *Data* of Table 4). Note that the perturbed samples are not included for the MNIST models because they are all benign. For `DeepSpeech-0.1.1`, the number of adversarial samples is 1,100, and we also select the same number of perturbed samples to make a fair comparison. For other models, we use all 10,000 perturbed/adversarial samples.

Table 4: Coverage criteria sensitivity to the slightly perturbed samples and adversarial samples.

Sub.	Conf.	Data	State (%)										Transition (%)					
mod.	(k, m)		BSCov		WSCov		WSCov'		3-SBCov		6-SBCov		BTCov		WTCov		WTCov'	
DS1	(3, 10)	*O*	67.9		99.7		3.0		0.0		0.0		23.5		95.5		23.5	
		O+P	74.5	10	99.9	0	3.2	7	0.0	0.0	0.0	0.0	37.9	61	98.7	3	37.9	61
		O+A	74.1	9	99.9	0	3.2	7	0.0	0.0	0.0	0.0	44.0	87	98.6	3	44.0	87
	(6, 10)	*O*	13.7		81.1		0.4		0.0		0.0		0.8		29.6		0.8	
		O+P	26.0	90	92.9	15	0.7	75	0.0	0.0	0.0	0.0	3.4	325	49.4	67	3.4	325
		O+A	35.4	158	94.0	16	0.9	125	0.1	0.1	0.0	0.0	6.4	700	53.0	79	6.4	700
	(6, 20)	*O*	1.7		22.9		0.1		0.0		0.0		0.1		5.1		0.1	
		O+P	7.2	324	47.1	106	0.4	300	0.0	0.0	0.0	0.0	0.4	300	9.1	78	0.4	300
		O+A	13.9	718	51.8	126	0.8	700	0.2	0.2	0.1	0.1	0.4	300	6.2	22	0.4	300
ML	(2, 5)	*O*	85.7		100.0		14.2		0.0		0.0		57.1		99.0		56.8	
		O+A	85.7	0	100.0	0	14.2	0	0.0	0.0	0.0	0.0	85.3	49	100.0	1	85.2	50
	(3, 5)	*O*	71.6		99.4		8.2		0.0		0.0		32.1		95.6		32.0	
		O+A	89.0	24	99.9	1	10.2	24	0.4	0.4	0.2	0.2	76.7	139	99.4	4	75.8	137
	(3, 10)	*O*	54.6		96.4		5.6		0.0		0.0		15.1		77.8		15.1	
		O+A	81.8	50	99.4	3	8.2	46	1.5	1.5	0.7	0.7	60.3	299	95.0	22	60.0	297

* The coverage increase ratio/value w.r.t. its seed group are marked with grey background.

Results. Table 4 reports the coverage results using different coverage criteria on different dataset (see results of `DeepSpeech-0.3.0` and `MNIST-GRU` on website [3]). The coverage increase ratio indicates the sensitivity of the coverage criteria to adversarial samples. We observed that finer-grained abstract models tend to have larger coverage increase ratio. This is because finer-grained state and transition information is more likely to distinguish adversarial samples from the benign samples. We also found that the sensitivities of various criteria to adversarial samples are rather different. For example, the increase ratio of WSCov/WTCov is relatively small because they mainly concern the frequently covered states and transitions (during profiling) which are often already fully covered by benign samples. In contrast, the increase ratio of WSCov'/WTCov' is larger, with a competitive performance as BSCov and BTCov, indicating rarely visited states and transitions tend to be covered by adversarial/perturbed samples. For the *n*-SBCov criteria, we present the *increase value* instead the increase ratio as the initial criteria are zero. The increase is not quite significant, as these states are really hard to cover even by adversarial samples. Furthermore, for `DeepSpeech-0.1.1`, we find that the coverage criteria of *O+A* are generally higher than those of *O+P*. This is possibly due to the larger average WER of the adversarial audios.

Answer to RQ2: The test coverage criteria are more sensitive with finer-grained abstraction. All proposed coverage criteria are sensitive to erroneous behaviors in adversarial samples, among which, BSCov, and BTCov are the most sensitive ones.

Table 5: AUROC results (%) of trace similarity based adversarial detection by configurations.

Config.	DeepSpeech-0.1.1		MNIST-LSTM					
(k, m)	STSim	TTSim	FSGM		BIM		DeepFool	
			STSim	TTSim	STSim	TTSim	STSim	TTSim
(2, 5)	49.78	67.67	77.97	79.8	74.61	74.85	75.11	73.55
(2, 10)	59.52	84.73	83.13	84.29	80.81	79.91	78.92	80.1
(2, 20)	80.90	81.24	83.37	84.71	80.42	80.42	79.09	80.27
(2, 40)	81.00	50.00	79.95	88.01	77.9	80.77	76.82	81.86
(2, 80)	70.25	50.00	90.54	**96.55**	82.38	**92.12**	84.43	**92.46**
(3, 5)	69.41	**85.40**	86.95	85.31	84.71	83.25	82.42	80.15
(3, 10)	**89.26**	**85.19**	90.05	89.74	**86.18**	85.28	**85.23**	83.78
(3, 20)	**85.25**	50.00	89.4	92.62	84.63	84.72	83.53	85.46
(3, 40)	50.49	50.00	**90.56**	**93.79**	85.97	86.62	84.05	87.47
(3, 80)	50.00	50.00	**96.63**	93.64	**92.97**	**89.75**	**93.44**	**90.43**
(6, 10)	75.82	50.00	89.81	86.68	85.18	78.04	84.95	80.59
(6, 20)	50.00	50.00	86.94	84.53	79.45	72.64	82.54	78.99
(6, 40)	50.00	50.00	85.94	92.17	78.64	85.08	79.96	87.79

6.4 RQ3. Adversarial Sample Detection

Setup. This section evaluates *DeepStellar* for adversarial sample detection on three models, namely, `DeepSpeech-0.1.1`, `MNIST-LTSM`, and `MNIST-GRU`. We first prepared the benign, adversarial, and reference samples (refer to B, A and R in Algorithm 1) to train the linear regression classifier. For the ASR model, we randomly selected 1,100 benign samples (800 for training and 300 for testing) from the test dataset, to make an equal number as the 1,100 generated adversarial samples. Specifically, the 11 target commands are divided into two sets C_1 and C_2, which contain 8 and 3 commands, respectively. The adversarial samples whose prediction results belong to C_1 are used as training data (*i.e.*, 100×8), while the other as test data (*i.e.*, 100×3). The reference samples are constructed by re-transcribing all the prediction results of both adversarial and benign samples to audios with the Google cloud text-to-speech [4]. For MNIST models, we take the 9,000 benign samples from the test data of MNIST and generate 9,000 adversarial samples with each of the three approaches, i.e., *FGSM*, *BIM* and *DeepFool*, of which 70% are used for training and 30% are used for testing. For reference samples, we randomly selected 50 samples from the training data of MINIST for each of the 10 categories (*i.e.*, 50×10). With the constructed dataset, we trained a classifier for each model to detect adversarial samples.

Results. Table 5 shows the AUROC [14] results of adversarial sample detection using different trace similarity metrics (*i.e.*, STSim and TTSim) and DTMC models with different configurations (Column *Config.*). The best two results of each model are highlighted, *e.g.*, 89.26% for `DeepSpeech-0.1.1`, 96.63% for MNIST-LSTM, and 96.63% for MNIST-GRU (see website [3]). The best results indicate that state-based trace similarity is a bit more effective than transition-based one in many cases. The results of MNIST models on detecting attacks generated by different tools show that our algorithm is robustness to a wide range of attacks, respectively with accuracy of 97%, 93%, and 93%. Furthermore, the results under DTMC models with different configurations vary largely. With finer-grained model, the result is not necessarily better. Overall, the results confirm that the trace similarity-based method is effective for adversarial sample detection under carefully selected abstraction configurations, with more than 89% prediction accuracy.

Table 6: Coverage and unique adversarial samples detected.

Criteria (%)	MNIST-LSTM					MNIST-GRU				
/Crash (#)	Seed	S-Guid.	T-Guid.	Ran.	DeepTest	Seed	S-Guid.	T-Guid.	Ran.	DeepTest
BSCov	54.59	**97.78**	**97.78**	86.23	65.35	63.20	95.87	**96.04**	88.78	68.98
WSCov	96.44	**99.99**	**99.99**	99.66	98.04	97.03	**99.98**	**99.98**	99.80	98.05
WSCov'	5.56	**9.97**	**9.97**	8.79	6.66	3.83	5.82	**5.83**	5.38	4.18
3-SBCov	0.00	1.50	**2.00**	1.86	0.07	0.00	0.78	**1.24**	0.85	0
6-SBCov	0.00	0.56	**0.75**	0.69	0.03	0.00	0.29	**0.46**	0.32	0
BTCov	15.13	53.43	**96.43**	73.88	26.23	14.42	42.80	**93.89**	71.82	21.32
WTCov	77.80	94.81	**99.90**	98.02	85.12	63.40	88.78	**99.69**	96.95	72.34
WTCov'	15.12	53.43	**96.43**	73.88	26.22	14.41	42.80	**93.89**	71.81	21.31
#Unique Cra.	-	**87,596**	41,614	2,219	300	-	**69,777**	35,228	19,738	244

* The last row presents the number of unique crashes discovered in each experiment.

Answer to RQ3: Similarity metric based method is useful for adversarial sample detection. The detection accuracy varies under different metrics and model configurations.

6.5 RQ4. Coverage-guided Testing

Setup. We use the prepared 100 original benign samples as the initial seeds, which are correctly predicted by both MNIST-LTSM and MNIST-GRU. Based on the results of RQ2, we use the fined-grained configuration (3,10) for constructing the DTMC models, and select BSCov and BTCov as the testing guidance. Finally, we implement two testing strategies, *i.e.*, *S-Guid.* and *T-Guid.*. To further demonstrate the usefulness of the coverage guidance, we include random testing without coverage guidance and *DeepTest* [50], a neuron coverage guided testing tool for unrolled RNNs, as baseline approaches for comparison. Each testing configuration was run for 6 hours, upon which the studied coverage criteria tend to saturate. To counter the randomness of testing tool, each configuration is repeated 5 times and averaged results are reported.

Results. Table 6 summarizes the obtained coverage results for different coverage criteria and the unique adversarial samples detected with the testing tools. The first column lists the studied coverage criteria. Column *Seed* represents the coverage of the initial seeds. Columns *S-Guid.*, *T-Guid.*, *Random* and *DeepTest* are the coverage by different testing strategies. We can observe that all of the studied strategies improve the coverage to some extent. Transition coverage-guided strategy outperforms the other two strategies in achieving higher coverage under all criteria. Furthermore, state coverage-guided strategy is often more effective in generating adversarial samples although it does not obtain the highest coverage. The overall results indicate that covering more new states could be potentially helpful in generating adversarial samples.

Answer to RQ4: The coverage-guided testing is generally useful in terms of achieving higher coverage and guiding adversarial sample exploration. Among the three strategies, transition coverage-guided method achieves higher coverage, while state coverage-guided method uncovers more unique adversarial samples.

6.6 Threats to Validity

We summarize factors that could affect the validity of our study. A major threat is related to the abstract model configuration settings.

There could be many possible configurations in the abstract model. Due to the computation resource constraint, we tried our best to experiment with as many settings as possible. Even though our results may still not generalize beyond the considered settings.

The subject model selection could be another threat to generalizability. We mitigate this by choosing models with diverse complexities and different application domains, covering both simple cases and industrial-grade ASR applications. In addition, for the ASR model, the training sets for the adversarial detection classifier is relatively small, which might affect the the performance of the detection. Adopting a larger training set may help obtain even better detection performance, which we leave as future work. Furthermore, randomness is a threat in both sampling, testing across our studied research questions. To counteract this, we repeat the same setting for all experiments five times and average the results.

7 RELATED WORK

In this section, we compare our work with other abstraction techniques, testing and adversarial sample detection for DL systems.

Abstraction of RNN. Several approaches have been proposed to model RNN, but mostly in the form of Finite State Automaton (FSA). FSA helps to manifest the internal state transitions explicitly and thus can be used to interpret the underlying decision rules embedded in an RNN. DTMC is superior with the ability to capture state transition distributions, making it more suitable for quantitative analysis. Constructing an FSA from an RNN usually requires two steps: (1) hidden state space partition and abstraction, and (2) transition abstraction and automaton construction. Various partitioning strategies and automaton construction algorithms have been proposed. Omlin and Giles [40] proposed to split each dimension of the state vector into equal intervals, so as to divide the state space into regular grids. Unsupervised classification algorithms were also applied for state space partitions. For example, k-means and its variants were studied in [9, 24, 53]. Weiss et al. [54] devised an algorithm to dynamically create partitions, where an SVM classifier with an RBF kernel is fitted to separate several state vectors from its original partitions. Recent studies [24, 53, 54] have focused more on the interpretability and visulizability of RNN behaviors, and try to simplify the abstract model by reducing the size of the models. When applied to real-world tasks, including NLP and speech recognition, the state space of the trained RNN models could be tremendously large. This makes scalability an issue for partition techniques such as k-means and kernel algorithms. However, we adopted a cheaper interval abstraction and could benefit from its flexibility in precision adjustment.

Testing of DNN. The lack of robustness places a major threat to the commercialization and wide adoption of DL systems. Researchers have devoted a great amount of efforts to investigate effective and systematic approaches to test DL systems, led with a pioneering work of Pei et al. [41]. The authors designed the first testing criterion – *neuron coverage* – to measure how much internal logic of DNNs has been examined by a given set of test data. Several new criteria have been proposed since then, including a set of multi-granularity testing criteria proposed in DeepGauge [32], a set of adapted MC/DC test criteria [45], and combinatorial testing criteria [31]. So far, the proposed coverage criteria are used to guide the metamorphic mutation-based testing [50], concolic testing [46], and coverage-guided testing of DNN [39, 55]. In addition, mutation testing technique is also proposed to evaluate the test data quality through injecting faults into DL models [33]. In [56], a black-box differential testing framework is proposed for detecting the disagreements between multiple models.

MC/DC criteria are limited in scalability, and other criteria are specific to the FNN architecture, even though applicable to RNN via unrolling. The results reported in [50] demonstrated that the neuron coverage works effectively on FNN but far from ideal on RNN when used to guide test generation. This indicates that RNN is beyond a simple folding of CNN, and existing criteria may not be well suited for it. Due to the page limit, we refer the interested readers to a comprehensive survey on machine learning testing [58].

Adversarial Sample Detection. Some techniques [15, 19, 29, 34, 52, 57] are proposed to detect adversarial samples that are predicted incorrectly. The authors found that adversarial samples are much more sensitive in model mutants, and proposed a sensitivity-based approach to detect adversarial samples for feed-forward models [52]. The technique in [36] augments the DNNs by adding a small subnetwork which obtains inputs from the intermediate feature representations of the DNN and is trained to detect adversarial samples. The authors proposed two features [15], *i.e.*, density estimates and Bayesian uncertainty estimates to show the differences between benign and adversarial samples. Xu et al. [57] adopt two types of feature squeezing, *i.e.*, reducing the color bit depth of each pixel and spatial smoothing to detect adversarial samples. These approaches mainly consider CNNs and the image classification domain.

In [19], the authors detect the adversarial samples based on that the adversarial samples have a relatively smaller softmax probability. Following this line, the technique proposed in [29] observes that the softmax probabilities between in- and out-of-distribution samples can be further enlarged by temperature scaling in the softmax function. Such methods can be used on RNNs but are limited to the classification problem. Compared with them, our approach can handle sequential outputs of RNNs (*e.g.*, the outputs in automated speech recognition) based on the abstract model.

8 CONCLUSION

Vulnerabilities of DL systems are threatening the trust and mass adoption of these technologies. This work initiates the first step towards the quantitative analysis of stateful DL systems. We model an RNN as an abstract model, based on which a set of similarity metrics and coverage criteria are proposed. We demonstrated the usefulness of the proposed models and quantitative measures on RNNs testing and adversarial sample detection. Our long term goal is to provide quality assurance for the DL system life-cycle [30].

ACKNOWLEDGMENTS

This research was supported (in part) by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001) administered by the National Cybersecurity R&D Directorate, and JSPS KAKENHI Grant 19H04086, and NTU research grant NGF-2019-06-024.

REFERENCES

[1] 2018. Mozilla Common Voice. https://voice.mozilla.org/en.

[2] 2018. Mozilla's DeepSpeech. https://github.com/mozilla/DeepSpeech.

[3] 2019. DeepStellar. https://sites.google.com/view/deepstellar/home

[4] 2019. Google cloud text-to-speech. https://cloud.google.com/text-to-speech/

[5] 2019. kaggle: Audio data augmentation. https://www.kaggle.com/CVxTz/audio-data-augmentation

[6] 2019. Levenshtein Distance. https://en.wikipedia.org/wiki/Levenshtein_distance

[7] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, and et al. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*. 265–283.

[8] Nicholas Carlini and David Wagner. 2018. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. (jan 2018). arXiv:1801.01944 http://arxiv.org/abs/1801.01944

[9] Adelmo Luis Cechin, Denise Regina Pechmann Simon, and Klaus Stertz. 2003. State Automata Extraction from Recurrent Neural Nets Using k-Means and Fuzzy Clustering. In *Proceedings of the XXIII International Conference of the Chilean Computer Science Society*. 73.

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[11] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

[12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *CVPR*. 3642–3649.

[13] Tianyu Du, Shouling Ji, Jinfeng Li, Qinchen Gu, Ting Wang, and Raheem Beyah. 2019. SirenAttack: Generating Adversarial Audio for End-to-End Acoustic Systems. *arXiv preprint arXiv:1901.07846* (2019).

[14] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[15] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410* (2017).

[16] Arthur Gill. 1962. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill. https://books.google.com.sg/books?id=2IzQAAAAMAAJ

[17] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. http://arxiv.org/abs/1412.6572

[18] Google Accident. 2016. A Google self-driving car caused a crash for the first time. https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report

[19] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[20] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[21] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[22] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural computation* 9, 8 (1997), 1735–1780.

[23] Bill G. Horne, C. Lee Giles, Pete C. Collingwood, School Of Computing, Man Sci, Peter Tino, and Peter Tino. 1998. Finite State Machines and Recurrent Neural Networks – Automata and Dynamical Systems Approaches. In *Neural Networks and Pattern Recognition*. Academic Press, 171–220.

[24] Bo-Jian Hou and Zhi-Hua Zhou. 2018. Learning with Interpretable Structure from RNN. (oct 2018). arXiv:1810.10708 http://arxiv.org/abs/1810.10708

[25] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. 2015. An Empirical Evaluation of Deep Learning on Highway Driving. *CoRR* abs/1504.01716 (2015). arXiv:1504.01716 http://arxiv.org/abs/1504.01716

[26] Ian Jolliffe. 2011. Principal Component Analysis. In *International Encyclopedia of Statistical Science*. Springer, 1094–1096.

[27] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. 1039–1049.

[28] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[29] Shiyu Liang, Yixuan Li, and R Srikant. 2017. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690* (2017).

[30] Lei Ma, Felix Juefei-Xu, Minhui Xue, Qiang Hu, Sen Chen, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. 2018. Secure Deep Learning Engineering: A Software Quality Assurance Perspective. *arXiv e-prints* (Oct. 2018), arXiv:1810.04538.

[31] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 614–618.

[32] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems. In *Proc. of the 33rd ACM/IEEE Intl. Conf. on Automated Software Engineering (ASE 2018)*. 120–131.

[33] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. [n. d.]. DeepMutation: Mutation Testing of Deep Learning Systems. In *29th IEEE International Symposium on Software Reliability Engineering (ISSRE), Memphis, USA, Oct. 15-18, 2018*. 100–111.

[34] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. 2019. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *NDSS*. 24–27.

[35] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[36] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267* (2017).

[37] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. *CVPR* (2016), 2574–2582.

[38] J. R. Norris. 1997. *Markov Chains*. Cambridge University Press. https://doi.org/10.1017/CBO9780511810633

[39] Augustus Odena and Ian Goodfellow. 2018. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. (2018). arXiv:1807.10875

[40] Christian W Omlin and C Lee Giles. 1996. Extraction of rules from discrete-time recurrent neural networks. *Neural networks* 9, 1 (1996), 41–52.

[41] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *SOSP*. 1–18.

[42] Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting Finite-State Transductions with Neural Context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 623–633.

[43] Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685* (2015).

[44] Charles Spearman. 1987. The proof and measurement of association between two things. *The American journal of psychology* 100, 3/4 (1987), 441–471.

[45] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *arXiv preprint arXiv:1803.04792* (2018).

[46] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. (2018). https://doi.org/arXiv:1805.00089v2 arXiv:1805.00089

[47] The BBC. 2016. AI image recognition fooled by single pixel change. https://www.bbc.com/news/technology-41845878

[48] The New York Times. 2016. Alexa and Siri Can Hear This Hidden Command. You Can't. https://www.nytimes.com/2018/05/10/technology/alexa-siri-hidden-command-audio-attacks.html

[49] Joe F Thompson, Bharat K Soni, and Nigel P Weatherill. 1998. *Handbook of Grid Generation*. CRC press.

[50] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *ICSE*. ACM, 303–314.

[51] Uber Accident. 2018. After Fatal Uber Crash, a Self-Driving Start-Up Moves Forward. https://www.nytimes.com/2018/05/07/technology/uber-crash-autonomous-driveai.html

[52] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2018. Adversarial Sample Detection for Deep Neural Network through Model Mutation Testing. *arXiv preprint arXiv:1812.05793* (2018).

[53] Qinglong Wang, Kaixuan Zhang, Alexander G. Ororbia, II, Xinyu Xing, Xue Liu, and C. Lee Giles. 2018. An Empirical Evaluation of Rule Extraction from Recurrent Neural Networks. *Neural Comput.* 30, 9 (Sept. 2018), 2568–2591.

[54] Gail Weiss, Yoav Goldberg, and Eran Yahav. 2017. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. *arXiv preprint arXiv:1711.09576* (2017).

[55] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *ISSTA*.

[56] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. 2019. DiffChaser: Detecting Disagreements for Deep Neural Networks. In *IJCAI*.

[57] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155* (2017).

[58] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2019. Machine Learning Testing: Survey, Landscapes and Horizons. *arXiv e-prints* (Jun 2019), arXiv:1906.10742.