DeepCruiser: Automated Guided Testing for Stateful Deep Learning Systems

Xiaoning Du[1], Xiaofei Xie[1], Yi Li[1], Lei Ma[2], Jianjun Zhao[3], and Yang Liu[1]

[1] Nanyang Technological University
[2] Harbin Institute of Technology
[3] Kyushu University

Abstract. Deep learning (DL) defines a data-driven programming paradigm that automatically composes the system decision logic from the training data. In company with the data explosion and hardware acceleration during the past decade, DL achieves tremendous success in many cutting-edge applications. However, even the state-of-the-art DL systems still suffer from quality and reliability issues. It was only until recently that some preliminary progress was made in testing feed-forward DL systems.In contrast to feed-forward DL systems, recurrent neural networks (RNN) follow a very different architectural design, implementing temporal behaviours and "memory" with loops and internal states. Such stateful nature of RNN contributes to its success in handling sequential inputs such as audio, natural languages and video processing, but also poses new challenges for quality assurance.

In this paper, we initiate the very first step towards testing RNN-based stateful DL systems. We model RNN as an abstract state transition system, based on which we define a set of test coverage criteria specialized for stateful DL systems. Moreover, we propose an automated testing framework, *DeepCruiser*, which systematically generates tests in large scale to uncover defects of stateful DL systems with coverage guidance. Our in-depth evaluation on a state-of-the-art speech-to-text DL system demonstrates the effectiveness of our technique in improving quality and reliability of stateful DL systems.

1 Introduction

Deep learning (DL) experiences significant progress over the past decades in achieving competitive performance of human intelligence in many cutting-edge applications such as image processing [1], speech recognition [2], autonomous driving [3], medical diagnosis [4] and pharmaceutical discovery [5], which until several years ago were still notoriously difficult to solve programmatically. DL has been continuously redefining the landscape of industry, in penetrating and reshaping almost every aspect of our society and daily life. For example, Automated Speech Recognition (ASR) currently becomes one of the most effective ways for human computer interaction and communication, and is widely integrated in intelligent assistants on everyday mobile device (e.g., Apple Siri [6], Amazon Alexa [7], Google Assistant [8], and Microsoft Cortana [9]). Although Hidden Markov Models (HMM) were widely used in ASR, DL models are the current state-of-the-art solutions in tasks including speech recognition and generation [10–12].

However, the current state-of-the-art DL system still suffers from quality, reliability and security issues, which could potentially lead to accidents and catastrophic events when deployed to safety- and security-critical systems. With the demanding industry trends for real-world deployment of DL solutions, we have witnessed many quality and security issues, such as one pixel attack [13], Alexa/Siri manipulation with hidden voice command [14], and the Google/Uber autonomous car accident [15, 16]. Unfortunately, the quality assurance techniques and tool chains for DL systems are still immature, which could potentially hinder future industry scale applications of DL solutions towards the goal of Software 2.0 [17].

For traditional software, the software development and quality assurance process are well established over past several decades, but the existing techniques and tool chains could not be directly applied to DL systems. This is mainly due to the fundamental differences in programming paradigms, development processes, as well as structures and logic representations of the software artifacts (e.g., architectures) [18, 19]. To bridge the gap between quality assurance of DL system and its practical applications, some recent work on testing and verification of feed-forward DL systems (e.g. Convolution Neural Network (CNN)) of image processing started to emerge, ranging from testing criteria design [18–20], test generation [21, 22], and metamorphic relation based testing oracle [23], to abstract interpretation based formal analysis [24].

Yet, such practices are hardly applicable to the testing of Recurrent Neural Networks (RNN) due to the very different architectural designs, often with loops involved, that introduces internal states and enables the "memorization" of what could be observed before or after [11]. Information flows not only from front neural layers to the rear ones, but also from the current iteration to the subsequent ones. This makes RNN more suitable to process sequential input streams, such as audios, natural language texts and videos, instead of monolithic data such as images. Although it is tempting to unroll the network and test the unfolded RNN as if it is a feed-forward neural network [23], such a simple approach ignores the internal states of RNNs and therefore could not precisely reflect the dynamic behaviors for a time sequence. Furthermore, the unrolled networks would contain different numbers of layers given inputs of various lengths, making the calculation of coverage problematic.

As a typical application of RNN, ASR is faced with the problem of inadequate testing, security threats and attacks [25–28]. Despite the urgent demands, it is unfortunate that there exists no systematic techniques specialized for RNN-based stateful deep learning systems at the moment. To address these challenges, in this paper, we propose a coverage-guided automated testing framework for RNN-based stateful DL systems. Considering the unique features of RNN and the input structures it often processes, we first propose to formalize and model a RNN-based DL system as Markov Decision Process (MDP). Based on the MDP model, we design a set of testing criteria specialized for RNN-based DL systems to capture its dynamic state transition behaviours. The proposed testing criteria enable the quantitative evaluation on how extensive the RNN's internal behaviors are covered by test data.

Based on this, we further propose an automated testing framework for RNN-based DL system guided by proposed coverage. In this paper, we focus on the ASR domain[4], and incorporate 8 metamorphic transformations to generate new audio test inputs. Due to the huge test input generation space, we leverage the coverage feedback to guide the testing direction towards systematically cover the major functional behaviors and corner cases of a RNN. We implement our testing framework, *DeepCruiser*, and perform in-depth evaluation on a state-of-the-art practical RNN based ASR system to demonstrate the usefulness.

To the best of our knowledge, this paper makes several novel contributions summarized as follows.

- We formalize a stateful DL system as a MDP, which is able to characterize the internal states and dynamic behaviors of RNN-based stateful DL systems.
- Based on the MDP abstraction, we design a set of specialized testing criteria for stateful DL systems, including two state-level criteria and three transition-level criteria. This is the very first set of testing criteria specially designed for RNNs.
- We evaluate the usefulness of the criteria on a real-world ASR application, and confirm that more precise abstraction can better discriminate different test sequences, and generating tests towards increasing coverage is helpful for defect detection.
- We implement a coverage-guided testing framework, *DeepCruiser*. As the first testing framework for audio-based DL systems, we also designed a set of metamorphic transformations tailored for audio inputs, inspired by real-world scenarios such as background noise, volume variation, etc. Experimental results demonstrate the effectiveness of *DeepCruiser* in terms of generating high-coverage tests and discovering defects on practical ASR systems.

The rest of this paper is organized as follows. Section 2 introduces the background of RNN-based stateful deep learning systems and one of its successful application domain, ASR, concerned in this paper. Section 3 presents the high-level workflow of our coverage-guided testing framework and tool implementation *DeepCruiser* for stateful DL systems. Section 4 discusses the detailed state transition modeling and formalization of RNN and Section 5 proposes the testing criteria specialized for RNN-based stateful DL systems. Based on these, Section 6 proposes a coverage-guided testing framework for defect detection in stateful DL systems. Section 7 performs a large scale evaluation of our proposed technique on a practical end-to-end RNN-based ASR system. Finally, we discuss the related work in Section 8 and concludes the paper with some possible future directions in Section 9.

2 Background

In this section, we introduce background of RNN and its applications in automated speech recognition.

[4] Although we focus on testing RNNs of ASR domain due to the currently urgent industry demands, the techniques and testing framework proposed in this paper can be generalized to other RNN-based stateful DL systems.

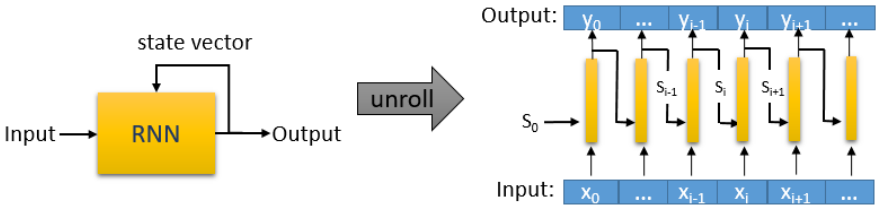


Fig. 1: Architecture of simplified RNN.

2.1 Recurrent Neural Network

Different from the feed-forward DL systems (e.g, CNN), Recurrent Neural Networks (RNNs) implement temporal behaviors with loops and memorization with internal states to take into account influence of previous (or future) observations. Such stateful nature of RNN contributes to its huge advantage and success in handling sequential inputs, and leads to its domination in current industrial applications on audio, natural languages and video processing, making huge impact on our daily life. A simplified version of the RNN architecture is illustrated in Figure 1. A basic RNN is a network of neuron-like nodes organized into successive *iterations* (or *loops*). It takes as inputs both the data stream to process 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, and sequentially produces outputs at each iteration and updates the state vector. For each individual input sequence, the state vector is first initialized to s_0 (usually a vector of zeros), and the state vector from previous iteration is passed to the next iteration. At the high level, the sequence of state vectors can be seen as a trace recording the underlying temporal dynamics of RNNs, thus providing a witness for the overall characteristics of the network.

With the loop design, the "vanishing gradient problem" [29] becomes more severe on RNN, where gradient used in back propagation of training can either vanish to zero or becomes extremely large when the number of iterations increases, causing the model difficult to be optimized. Long Short-Term Memory (LSTM) [30] and Gated Recurrent Unit (GRU) [31] are designed to overcome this problem. Their network structures are much more complicated than the basic RNN shown in Figure 1, but all shares the simple basic principle of RNN design and implements the state vectors for memorization.

2.2 Automated Speech Recognition

ASR plays an important role in intelligent voice assistants, e.g., Amazon Alexa, Google Assistant and Apple Siri. The essential component of an ASR is a *transcribing module* responsible for converting speech features to texts. Historically, developing the transcribing module requires significant manual efforts in deriving the phonetic alignment of audios and texts, which severally limits the scale of the training data. With advanced training loss functions, such as "Connectionist Temporal Classification" (CTC) loss [32], the alignment can be handled internally by an end-to-end approach. The transcribing modules in current end-to-end ASRs are mostly equipped with a RNN kernel which may also be accompanied by some auxiliary CNN layers. Most popular open source

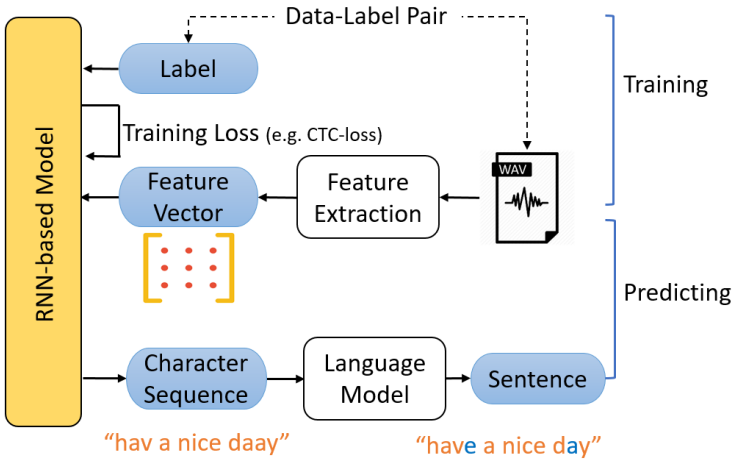


Fig. 2: General training and predicting workflow of end-to-end ASR systems.

end-to-end ASR systems, including DeepSpeech [11] and EESEN [33], are powered by RNN-based models.

A general workflow of the end-to-end ASR system is demonstrated in Figure 2, with training and predicting procedures highlighted, respectively. These two procedures share the same *feature extraction module*, where a raw audio is transformed to numerical features (e.g., Mel-Frequency Cepstral Coefficients (MFCC) features). During the training stage, feature vectors and text transcriptions are fed directly into the RNN-based model, skipping the acoustic model for phonetic analysis. As for prediction, the pipeline is composed of three major components, a *feature extraction module*, a *RNN-based translation module* and a *language model-based correction module*. Usually, the *language model* is provided externally and requires no training. Audios are first transformed into numerical feature vectors, and then passed to the DNN model to obtain initial transcriptions, which can be further refined or corrected by the *language model*. For example, in Figure 2, the spelling errors in the character sequence are easily rectified.

3 The Overview of Coverage-guided Testing Framework *DeepCruiser*

Figure 3 summarizes the workflow of our approach, including the *abstract model construction* of RNN, the *coverage criteria* defined over the RNN model, and a *coverage-guided automated testing framework* to generate tests for defect and vulnerability detection of RNNs.

The abstract model construction module takes a trained RNN as input and analyzes its internal runtime behaviors through profiling. The quality of the profiling largely depends on the input data used. The ideal choice of inputs for profiling is the training data (or part of them), which best reflect the internal dynamics of a trained RNN model. Specifically, each input sequence is profiled to derive a *trace*, i.e., a sequence of RNN

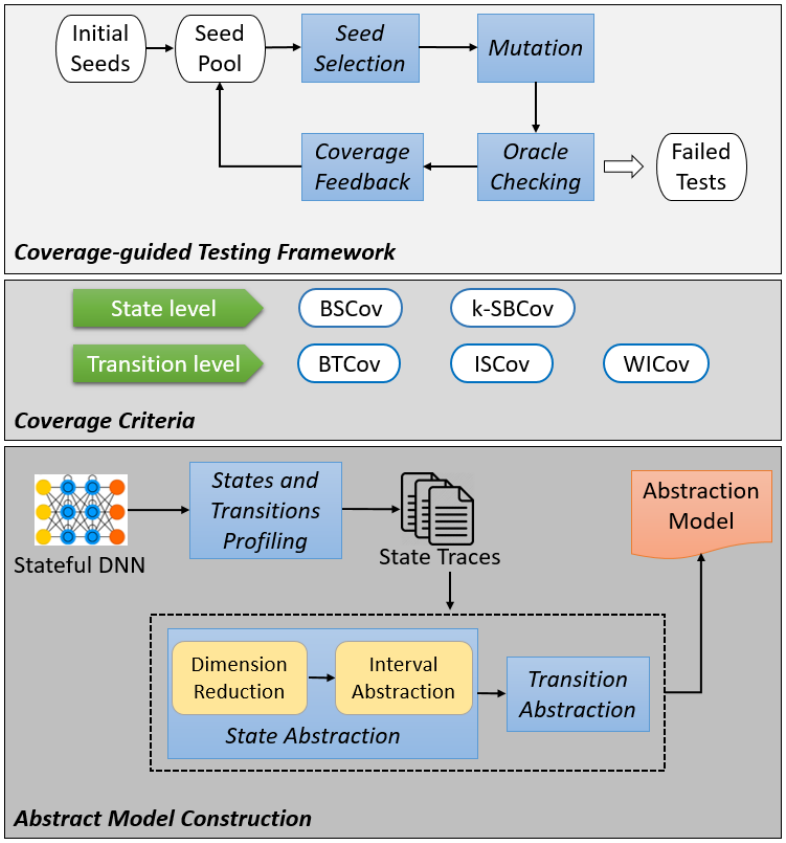


Fig. 3: High-level workflow of *DeepCruiser*.

state vectors. After all the provided inputs are profiled, we will get a set of traces which describe the major states visited and transitions taken by an RNN.

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 capturing the global characteristics of the trained network. At the state level, we apply Principle Component Analysis (PCA) [34] to reduce the dimensions of the vectors and keeps the first k dominant components. For each of the k dimensions, we further partition them into m equal intervals. At the transition level, we consolidate concrete transitions into abstract ones according to the state abstraction. We also take into account the frequencies of different transitions under various inputs and effectively derive a Markov Decision Process (MDP) [35] model for the trained RNN.

Based on the abstract model, five coverage criteria are then designed to facilitate the systematic testing of RNN. These include two state-level coverage criteria – the *basic state coverage* (BSCov) and the *k-step state boundary coverage* (k-SBCov), and

three transition-level criteria – the *basic transition coverage* (BTCov), the *input space coverage* (ISCov) and the *weighted input coverage* (WICov). These criteria are used to guide test case generation in our evaluation with the aim to uncover defects in the RNN under test (see Section 7).

The testing framework is designed to make use of the above criteria to facilitate the defect discovery of RNN-based systems. With a chosen coverage criterion, the testing process starts with a set of initial seeds (some audios under practical ASR testing scenarios), namely the *seed pool*. Then in each iteration, we select an audio based on heuristics and generate a mutant from it by applying certain transformations. If the generated mutant triggers some defects, e.g., the transcription shows a large difference from the expectation, it is labeled as a failing test. Otherwise, we check whether the mutant improves the test coverage with respect to the chosen criteria and include it into the seed pool if so.

4 State Transition Modeling of Recurrent Neural Network

RNN models are inherently stateful [36]. In this section, we formalize the internal states and state transitions of RNNs, and describe an abstract model used to capture the global characteristics of the trained RNN models.

4.1 RNN Internal States and State Transitions

Following [37], we represent a neural network abstractly as a differentiable parameterized function $f(\cdot)$. The input to a 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 that sequence. Then, when passing $\mathbf{x}$ into a 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) [38] can be used to represent a collection of traces more compactly [39] as defined below.

Definition 1. *A 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. 4 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 $y_0 y_1' y_2$.

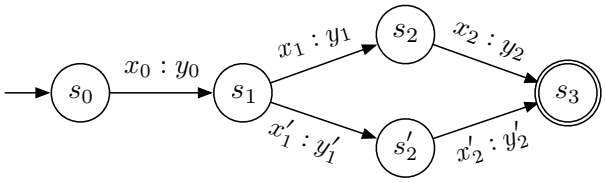


Fig. 4: An example FST representing two traces.

4.2 Abstract State Transition Model

The number of states and traces enabled while training a 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) [34] 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.

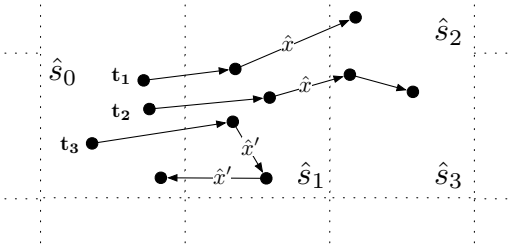
Then, we split the new k-dimensional space into m^k *regular grids* [40] such that there are m equal-length intervals on each axis: where e_i^d represents the i-th interval on the d-th dimension, lb_d and ub_d are the lower and upper bounds of all state vectors on the d-th dimension, respectively. In 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 $(0 \leq j < m)$. For any two abstract states $\hat{s}$ and $\hat{s}'$, we define their *distance* as:

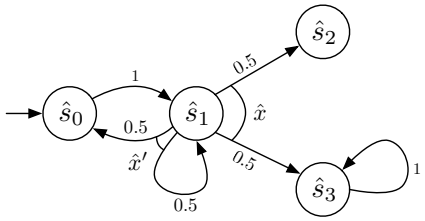
$$Dist(\hat{s}, \hat{s}') = \Sigma_{d=1}^{k} |I^d(\hat{s}) - I^d(\hat{s}')|.$$

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

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



(a) Concrete traces.



(b) MDP abstraction.

Fig. 5: A set of concrete traces and their corresponding abstract state transition model.

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, Figure 5a 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 connecting dots. 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, $\{(\hat{s}_0, \hat{s}_1), (\hat{s}_1, \hat{s}_0), (\hat{s}_1, \hat{s}_1), (\hat{s}_1, \hat{s}_2), (\hat{s}_1, \hat{s}_3), (\hat{s}_3, \hat{s}_3)\}$.

4.3 Representing Trained RNN as a Markov Decision Process

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. Because of the ways how the state and transition abstractions are defined, the resulting abstract model represents an over-approximation and generalization of the observed behaviors of the trained RNN model.

To also take into account the likelihood of abstract transitions under different inputs, we augment the abstract model with transition probabilities and non-deterministic choices, effectively making it a Markov Decision Process (MDP) [35] without costs.

Definition 2. *A MDP 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 $\hat{\mathcal{T}} : \hat{\mathcal{S}} \times \hat{\mathcal{X}} \mapsto Dist(\hat{\mathcal{S}})$ is the* transition probability function *such that $\hat{\mathcal{X}}$ represent the (abstract) input space and $Dist(\hat{\mathcal{S}})$ is the set of discrete probability distributions over the set of abstract states.*

The space of choices at a state $\hat{s}$ is given by the set of inputs $\hat{\mathcal{X}}(\hat{s})$ enabled at that state, which is abstracted in the same way as the states. We write $\Pr_{\hat{x}}(\hat{s}, \hat{s}')$ to denote the conditional probability of visiting $\hat{s}'$ given the current state $\hat{s}$ with input $\hat{x}$, such that $\Sigma_{\hat{s}' \in \hat{\mathcal{S}}} \Pr_{\hat{x}}(\hat{s}, \hat{s}') = 1$. 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 $\hat{s}$ given the input $\hat{x}$, i.e.,

$$\Pr_{\hat{x}}(\hat{s}, \hat{s}') = \frac{|\{(s, s', x) | s \in \hat{s} \wedge x \in \hat{x} \wedge s' \in \hat{s}'\}|}{|\{(s, \_, x) | s \in \hat{s} \wedge x \in \hat{x}\}|}.$$

For example, Figure 5b shows the abstract state transition model for the concrete traces in Figure 5a as a MDP. 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 one. There are two choices of inputs at $\hat{s}_1$, i.e., $\hat{\mathcal{X}}(\hat{s}_1) = \{\hat{x}, \hat{x}'\}$. When the given input at $\hat{s}_1$ is $\hat{x}$, the transition probabilities are computed as $\Pr_{\hat{x}}(\hat{s}_1, \hat{s}_2) = \frac{1}{2}$ and $\Pr_{\hat{x}}(\hat{s}_1, \hat{s}_3) = \frac{1}{2}$. The computation for the case when the input is $\hat{x}'$ is similar.

As is shown in the example, a MDP 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 input at every 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.

5 Coverage Criteria of Stateful Recurrent Neural Network

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 completeness and thoroughness 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 MDP. 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.

5.1 State-Level Coverage Criteria

The state-level coverage criteria focuses 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* [19]. The space outside the major function region is never visited by the training data, and thus represents the *corner-case region* [19]. 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 a 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:

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

k-Step State Boundary Coverage. The test data may also trigger new states which are never visited during training. The *k-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 *k-step boundary region*, $\hat{\mathcal{S}}_{M^c}(k)$, contains all abstract states which have a minimal distance k from $\hat{\mathcal{S}}_M$, or more formally, $\hat{\mathcal{S}}_{M^c}(k) = \{\hat{s} \in \hat{\mathcal{S}}_{M^c} | \min_{\hat{s}' \in \hat{\mathcal{S}}_M} Dist(\hat{s}, \hat{s}') = k\}$.

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

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

5.2 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. Then the *basic transition coverage* is given by:

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

The basic transition coverage subsumes the basic state coverage. In other words, for any abstract model M, every test input T satisfies basic transition coverage with respect to M, also satisfies the basic state coverage.

Input Space Coverage. The input space at an abstract state $\hat{s}$ is given as $\hat{\mathcal{X}}_{\hat{s}}$, which represents the set of abstract inputs accepted at $\hat{s}$ while training. The test inputs should

cover the input space at each state as much as possible to exercise the different subsequent transitions. More formally, let the input spaces for the training and test data at $\hat{s}$ be $\hat{\mathcal{X}}_M(\hat{s})$ and $\hat{\mathcal{X}}_T(\hat{s})$, respectively. Then the *input space coverage* is defined as:

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

Note that the input space coverage is incomparable to the basic transition coverage – achieving the input coverage does not guarantee the transition coverage, and vice versa.

Weighted Input Coverage. Our abstract model also encodes the frequencies of different transitions given a particular input, observed during training. More frequently triggered transitions have a higher transition probability, which is given by the transition probability function $\hat{\mathcal{T}}$. The *weighted input coverage* considers not only the different choices at a state, but also the range of possible subsequent transitions when a specific input is chosen. More formally, it is defined as:

$$\text{WICov}(T, M) = \frac{\Sigma_{\hat{s} \in \hat{\mathcal{S}}_T \cap \hat{\mathcal{S}}_M} \Sigma_{\hat{x} \in \hat{\mathcal{X}}_T(\hat{s}) \cap \hat{\mathcal{X}}_M(\hat{s})} \Sigma_{\hat{s}' \in \hat{\delta}_T(\hat{s})} \hat{\mathcal{T}}(\hat{s}, \hat{x}, \hat{s}')}{\Sigma_{\hat{s} \in \hat{\mathcal{S}}_M} |\hat{\mathcal{X}}_M(\hat{s})|}$$

The weighted input coverage is stronger than both the basic transition coverage and the input space coverage.

6 Coverage-Guided Testing Framework

In this section, we introduce the coverage-guided testing framework. We first describe a group of metamorphic transformations specialized for audio signals (Section 6.1) and then present a mutation-based test generation algorithm which is guided by the coverage criteria proposed in Section 5.

6.1 Metamorphic Transformations of Audio Signals

ASR performs general transformation from audio speeches to the corresponding natural-language texts, and is often expected to work properly on speeches with various volume, speed and voice characteristics. Also, these speech audio signals may be mixed with noises coming from ambient sounds and not well-insulated receivers. Inspired by these practical scenarios, we derive a set of transformation operators to mimic the environment interference. Overall, they can be classified into four categories:

- *Volume-related transformations (VRT):* ChangeVolume, LowPassFilter, HighPassFilter.
- *Speed-related transformations (SRT):* PitchShift, ChangeSpeed.
- *Clearness-related transformations (CRT):* AddWhiteNoise, ExtractHarmonic
- *Unaffected transformations (UAT):* DRC, Trim.

Table 1: Transformations for audio signals.

Transformation	Description
1 *AddWhiteNoise*	Randomly add white noise in the audio
2 *PitchShift*	Pitch-shift the waveform of the audio to raise or lower the pitch of an audio signal by a random interval
3 *Trim*	Trim leading and trailing silence from an audio signal
4 *ChangeSpeed*	Randomly speed up or slow down the audio
5 *ChangeVolume*	Randomly adjust the volume of the audio
6 *DRC*	Dynamic range compression (DRC), that reduces the volume of loud sounds or amplifies quiet sounds thus reducing or compressing an audio signal's dynamic range
7 *LowPassFilter*	Pass signals with a frequency lower than a random selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency
8 *HighPassFilter*	Pass signals with a frequency higher than a random cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency

Categories *VRT, SRT* and *CRT* affect the volume, speed and clearness of an audio signal, respectively; and Category *UAT* contains transformations that affect neither of them, but still makes minor changes to the speech signal. Table 1 summarizes the transformations with brief descriptions.

For defect detection, our goal is to generate audios which sounds normal to human but are incorrectly transcribed by ASRs. With a diverse collection of transformations, an audio can be mutated to generate new audios, among which there could be ones trigger new traces in the trained network and lead to potential defects in the ASR. However, a violent transformations with significant perturbations may result in an audio which is not recognizable even by human. For instance, the volume may become too low or the frequency may become too high. To generate suitable defect-triggering candidates, we apply the transformations with restraints to ensure that the audio seeds and the corresponding mutants sound the same to human beings. Transformation operators satisfying the above requirements are said to keep a *metamorphic relation* [41] and we refer to them as *metamorphic transformations*. Now we propose a strategy to perform transformations made by *DeepCruiser* with the best effort to preserve its text information of the audio before and after transformation.

In general, volume, speed and clearness transformations have no effect on the audio semantics, when applied individually and controlled under a certain degree. However, when an audio input a_0 is processed with a sequence of transformations (e.g., $a_0 \hookrightarrow a_1, \ldots, \hookrightarrow a_i$), there is a higher chance that the audio cannot be well recognized by human due to accumulated distortions. We propose a strategy to conservatively select

transformations for generating defect candidates indistinguishable to human before and after the transformations: (1) we carefully set the parameters to ensure that a single step transformation (on volume, speed or clearness) does not violate the metamorphic relations (i.e., human hearing are not affected); and (2) an audio, generated by mutating from a seed after a sequence of transformations T, is limited to be mutated by a transformation $t \in S \cup UAT$ such that $(\{t\} \cup S) \cap T = \emptyset$, where $S \in \{VR, SRT, CRT\}$. Intuitively, the constraints make sure that a mutant is generated by altering the volume, speed or clearness of a seed input at most once. If this constraints is unsatisfied, the audio will not be transformed further.

6.2 Coverage-Guided Testing

Algorithm 1 presents the general procedure to test RNN-based DL systems with various configurable feedback. It is consistent with the testing framework diagram shown in Fig. 3. The inputs of *DeepCruiser* include initial seeds I and the RNN-based deep learning system R. The outputs are regression tests with higher coverage and failed tests that are incorrectly handled. The initial test queue only contains initial seeds. For each time, *DeepCruiser* selects one input a (i.e., an speech audio) from the test queue. Based on the already selected transformations of a, *DeepCruiser* randomly picks one transformation t (c.f. Section 6.1). If no further transformation is allowed to be picked (i.e., $t = null$) under the metamorphic relation constraints, *DeepCruiser* will select next input from the queue. For a picked transformation t, a random parameter is picked. *DeepCruiser* will generate a new audio a' with transformation t and perform the transformation with the deep learning system R. If the prediction result is inconsistent with the original seed, a' will be added into the failed tests. For audio in ASR, we decide a failed test by checking whether the Word/Character Error Rate (W/CER) exceeds a certain threshold. If the mutated test is not a failed test and increases the overall coverage, *DeepCruiser* adds it into the test queue and updates the coverage of tests in the test queue.

7 Evaluation

In this section, we evaluate the effectiveness of the proposed abstract state transition model and test coverage criteria, and the performance of our coverage-guided testing framework. Through various experiments, we aim to answer the following research questions:

RQ1: Could the abstract model distinguish internal behaviours of RNN when handling different inputs? How precise is the distinction with different abstraction configurations?

RQ2: Is there a correlation between the proposed criteria and erroneous behaviors of the RNN?

RQ3: How effective is *DeepCruiser* for generating high-coverage tests?

RQ4: How useful is *DeepCruiser* for defect detection in RNN-based ASR systems?

Algorithm 1: *DeepCruiser*

input : I: Initial seeds, R, RNN-based stateful deep learning system
output : F: Failed tests, Q: Test queue

$F \leftarrow \emptyset$;
$Q \leftarrow I$;
while $a \leftarrow Select(Q)$ **do**
  $t \leftarrow PickTransform(a)$;
  if t *is null* **then**
    continue :;
  Randomly pick parameter p for t ;
  a' = mutate(t, a)vt;
  $cov, result \leftarrow Predicate(R, a')$;
  if *Failed(a', result)* **then**
    $F \leftarrow F \bigcup \{a'\}$
  else if *CoverageIncrease(cov, Q)* **then**
    $Q \leftarrow Q \bigcup a'$;
    $UpdateCoverage(Q)$;

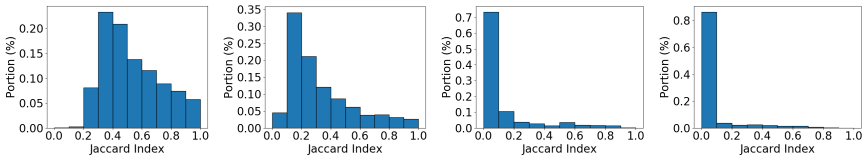
Table 2: Configurations of abstract model.

Name	$M_{\{3,10\}}$	$M_{\{3,20\}}$	$M_{\{3,50\}}$	$M_{\{3,100\}}$
#Dimensions (k)	3	3	3	3
#Partitions (m)	10	20	50	100
#States ($\lvert\hat{\mathcal{S}}\rvert$)	553	3,291	40,589	263,106

7.1 Datasets and Experiment Setup

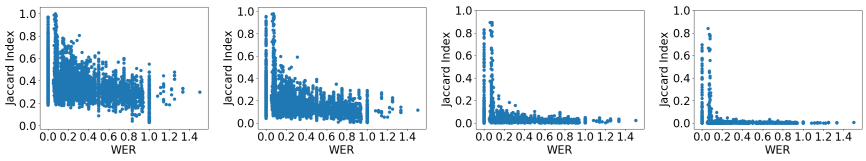
Datasets. We selected Mozilla's implementation of `DeepSpeech-0.3.0` [42] which produces one of the state-of-the-art open source ASR models. All our experiments were conducted on a pre-trained English model released along with `DeepSpeech-0.3.0`, which was trained with Fisher [43], LibriSpeech [44], Switchboard [45], and a pre-release snapshot of the English Common Voice training corpus [46]. It achieves an 11% WER on the LibriSpeech clean test corpus. In their implementation, the RNN-based model (see the part highlighted in Fig 2) is specialized with a LSTM kernel. Its state vector has 2048 dimensional 64-bit floating point, which is the major target for abstract model construction.

Experiment Setup. To construct the abstract model, we randomly selected 20% of the training data to perform the profiling considering our limited computation resource. Even though, after profiling, we still get a huge set of state vectors (about 90 billion 64-bit floats). In the next step, we perform Principle Component Analysis to analyze the principle components of the vector space, based on which the abstract model and transition space are constructed.



(a) Under $M_{\{3,10\}}$ (b) Under $M_{\{3,20\}}$ (c) Under $M_{\{3,50\}}$ (d) Under $M_{\{3,100\}}$

Fig. 6: Results of input similarities under different abstraction configurations.



(a) Under $M_{\{3,10\}}$ (b) Under $M_{\{3,20\}}$ (c) Under $M_{\{3,50\}}$ (d) Under $M_{\{3,100\}}$

Fig. 7: Correlation between basic state variation and relative WER under different abstraction configurations.

The model abstraction parameters k and m for the can be configured to generate abstract models with different precision. To analyze the potential influence on these parameters on model precision, we select four different configurations listed in Table 2. We use $M_{\{k,m\}}$ to represent the configuration with k dimensions and m partitions. The last row shows the number of states under different configurations.

7.2 RQ1: Evaluation on Abstract Model Precision

Since our testing criteria are designed based on the abstract model, the precision of the abstract model would directly influence the precision of the testing criteria in distinguishing the internal behaviours of RNN given different inputs. An accurate parameter configuration of testing criteria would allow to distinguish on the internal behaviours of RNN on even highly similar inputs. To generate new tests, we perform metamorphic transformations (Section 6.1) with very small changes to keep the metamorphic constraints with the best effort. Specifically, in this evaluation, we randomly select 100 inputs from the test data. For each audio, 100 inputs are generated using different metamorphic transformations. Finally, we have 100 original input and 10,000 new inputs.

To quantify the similarity between two inputs, we adopt the Jaccard index to measure their coverage similarity. Given an abstract model M and an instance x, we denote the set of basic states covered by x as $\hat{\mathcal{S}}_x$. Then the Jaccard Index $\mathrm{J_M}(x, y)$ for inputs x and y is calculated as:

$$\mathrm{J_M}(x, y) = \frac{|\hat{\mathcal{S}}_x \cap \hat{\mathcal{S}}_y|}{|\hat{\mathcal{S}}_x \cup \hat{\mathcal{S}}_y|}$$

Jaccard Index ranges over $[0, 1]$, and 0 indicates no overlapping (i.e., x and y is totally different) between two sets while 1 for totally duplicate sets (i.e., x and y is very similar).

Note that when Jaccard Index is 1, it does not imply that x and y are absolutely the same as the transitions may be different.

For each one of the new inputs and the corresponding original input, we compute a Jaccard Index value. Figure 6 shows the distribution of the 10,000 Jaccard Index values under different abstraction configurations. Under the configuration $M_{\{3,10\}}$, some of inputs cannot be be well differentiated as the configuration is too coarse. For exmaple, more than 90% of Jaccard Index values are greater or equal than 0.3, and even more than 5% values are distributed in the range $[0.9, 1.0]$. After the abstraction is refined (i.e., the grids are more fine-grained), most of the Jaccard Index values become smaller. For example, under $M_{\{3,100\}}$, more than 85% of values are distributed in the range $[0.0, 0.1]$.

Answer to RQ 1: The abstract model can distinguish internal behaviors of RNN for different inputs effectively, even for the inputs with small differences (e.g., the inputs with metamorphic relations). The accuracy of abstract models under different abstraction configurations also varies. Abstraction with more fine-grained grids distinguishes the outputs better.

7.3 RQ2: Correlation Between Testing Criteria and Erroneous Behaviors of the RNN

With RQ1, we already know that the minor differences in the inputs can be captured by the abstract model. Based on the abstract model, we proposed diverse testing criteria. In this section, we aim to evaluate whether the testing criteria can help to find potential defects and issues in RNNs, i.e., is there some correlation between testing coverage[5] and erroneous behavior of the RNN?

We sample 100 audio from the test data whose *Word Error Rates* (WERs) are 0. The WER of the audio represents the error rate for the transcripts from RNN. In other words, the sampled audio could be perfectly processed by the RNN. Based on the 100 audio (seeds), we randomly generate 10,000 audio (with different WER) by metamorphic transformations.

Figure 7 shows the distribution of the 10,000 audio under different abstraction configurations. The x-axis is WER and the y-axis is the Jaccard Index value. The results show that, in general, test cases with higher relative WER tend to have lower Jaccard index, which means they are less similar with the original seed input. The more fine-grained the abstract model is, the more obvious is such phenomenon. Intuitively, by increasing the state coverage, we can generate more data which have lower Jaccard index. Thereby attempting to increase the coverage on state offers more possibility to detect more erroneous behaviors of the RNN.

[5] In this paper, we focus on the *basic state coverage*, leaving the evaluation of other criteria in the future work.

Table 3: Basic state coverage increase with *DeepCruiser*.

Config.	Ini. Cov.	12h Cov.	Increase
$M_{\{3,10\}}$	26.1	43.2	65.5%
$M_{\{3,20\}}$	20.1	40.5	101.5%
$M_{\{3,50\}}$	12.3	36.2	194.3%
$M_{\{3,100\}}$	5.5	28.4	416.4%

Answer to RQ 2: There is a strong correlation between the state variation (i.e., the Jaccard Index) and erroneous behaviors of the RNN (i.e., WER). The transformed audio is likely to trigger more erroneous behaviors if it covers more different states comparing with the original audio. By improving BSC coverage, more states are covered and more erroneous behaviors would potentially be captured.

7.4 RQ3: Effectiveness of *DeepCruiser* for coverage improvement

DeepCruiser is designed to generate test cases with high coverage based on the coverage feedback. We evaluate the effectiveness of *DeepCruiser* in generating high-coverage tests with BSC guidance. Experimentally, we sample 100 audio as the initial seeds and run *DeepCruiser* for 12 hours. Table 3 shows the increase of BSC with different abstraction configurations. Columns "Ini. Cov." and "12h Cov." present the initial coverage of seeds and the coverage achieved after 12 hours' testing, respectively. Column "Increase" shows the coverage increases with respect to the initial values. The results show that *DeepCruiser* can increase the coverage effectively. As the configuration becomes more and more fine-grained (from $M_{\{3,10\}}$ to $M_{\{3,100\}}$), the initial coverage is smaller because the states are more in the abstract model. At the same time, the increment of the coverage becomes larger.

Answer to RQ 3: *DeepCruiser* can obviously improve the state coverage. Furthermore, it is more effective when the abstract model is more fine-grained.

7.5 RQ4: Erroneous Behavior Detection

To answer the question, we measure the WER of the generated audio in Fig. 8. WER represents the erroneous degree of the RNN prediction. Fig. 8 shows the average WER of generated audio under different abstraction configurations. The results show that the average WER is higher for more fine-grained abstract model. It can be explained by the answer to RQ2. With more fine-grained abstract model, *DeepCruiser* will generate test cases that can cover more states of the model. Thus, the test cases are more different with the original seed input (i.e., the Jaccard Index is smaller). Finally, it is more likely to generate test cases with higher WER.

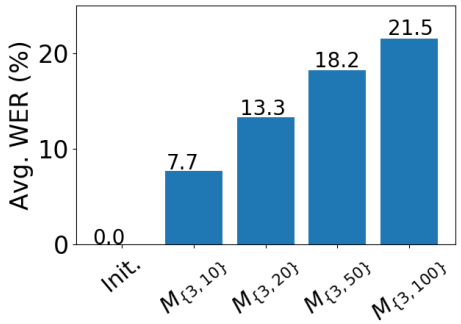


Fig. 8: Average WER of inputs generated with *DeepCruiser*.

Answer to RQ 4: *DeepCruiser* can effectively generate tests to trigger erroneous behaviors of the RNN. For more fine-grained abstract model, it will capture more erroneous behaviors (i.e., higher WER) of the RNN.

7.6 Threats to Validity

We list factors which could affect the validity of the experiments. Due to resource constraints, we did not use all of the training data for constructing the abstract model. This may result in a abstract model which does not fully reflect the actual trained network behaviors. To mitigate this problem, we randomly select samples from the training set and follow the distribution of training data approximately. We adopt a conservative metamorphic transformation strategy to make small changes on the original audio such that *DeepCruiser* will generate realistic audio. However, it may still cause false positives especially for the low-quality input (i.e., the audio is not clear such as low volume, too much noise). To mitigate this problem, we manually check and make sure the selected inputs (i.e., the 100 audio in RQ1, RQ2, RQ3 and RQ4) are of high quality.

8 Related Work

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

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 by the pioneer work of Pei et al. [18]. The authors designed the first set of testing criteria – *neuron coverage* – to measure how much internal logic of the DNN has been examined by a given set of test data. Several new criteria have been proposed since then, including a set of adapted MC/DC test criteria by Sun et al. [47], a set of multi-granularity testing criteria by Ma et al. [19], and combinatorial testing criteria [48]. So far, the proposed coverage criteria are used to guide the metamorphic mutation-based testing [23], concolic

testing [49], and coverage-guided testing of DNN [50, 51]. In addition, mutation testing techniques are also proposed to evaluate the test data quality through injecting faults into DL models [52].

The usefulness of MC/DC criteria is limited by the scalability issue, and other criteria are more suitable for the feedforward neural network architecture, even through they are partially applicable to RNN via unrolling. The experimental results reported in [23] demonstrated that the neuron coverage works effectively on CNN but far from ideal on RNN when used to guide the generation of test cases. This indicates that RNN are not simple folding of CNN, and existing criteria may not be well suited for RNN. Currently, there is still a lack of customized testing criteria specially designed for RNN, which are able to capture the statefulness of RNN and measure the thoroughness of the testing efforts.

Abstraction of RNN. Many approaches have been proposed to model RNN, usually in the form of Finite State Automaton (FSA), in order to understand the internal dynamics of RNN. FSA represents the internal state transitions explicitly and thus can be used to interpret the underlying decision rules embedded in RNN. MDP has similar properties in that sense, and it also captures state transition distributions under different inputs, making it a more precise model. Constructing a FSA from RNN requires two steps: (1) state space partition over the real-valued numerical vectors; and (2) automaton construction based on the partitions. Various partitioning strategies and automaton construction algorithms have been proposed. Omlin and Giles [53] proposed to divide 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 [54–56]. Weiss et al. [24] 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 [24,55,56] have focused more on the LSTM and GRU, demonstrating that the same abstraction techniques also work on RNN variants. When applied to real-world tasks, including natural language processing and speech recognition, the state space of the trained RNN model could be tremendously large. This makes scalability an issue for partition techniques such as k-means and kernel algorithms. Therefore, we adopted the much cheaper interval abstraction and we could also benefit from its flexibility in precision adjustment.

9 Conclusion

Vulnerabilities of current DL systems, such as autonomous vehicles and voice assistants, are threatening the trust and mass adoption of these technologies. As such, much research efforts have been focused on the testing of DL systems to ensure their reliability and robustness. Yet, little work has been done on the testing of stateful DL systems. As the first work along this line, we designed a set of test coverage criteria, which can be used to guide the systematic testing of software systems powered by stateful neural networks.

Furthermore, we proposed a set of metamorphic transformations on audios inspired by real-world scenarios, and implemented a general fuzzing framework to discover defects in ASRs. We confirmed the usefulness of the proposed criteria on ASRs and

showed that the fuzzing framework is effective in exposing real-world defects. In the future, we plan to evaluate our techniques on more diverse application domains, towards providing quality assurance solution for DL systems life-cycle [57].

References

1. D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *CVPR*, 2012, pp. 3642–3649.
2. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
3. B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving," *CoRR*, vol. abs/1504.01716, 2015. [Online]. Available: http://arxiv.org/abs/1504.01716
4. D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images," in *NIPS*, 2012, pp. 2843–2851.
5. H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The Rise of Deep Learning in Drug Discovery," *Drug Discovery Today*, vol. 23, no. 6, pp. 1241 – 1250, 2018.
6. "Apple Siri." [Online]. Available: https://www.apple.com/ios/siri
7. "Amazon Alexa." [Online]. Available: https://developer.amazon.com/alexa
8. "Google Assistant." [Online]. Available: https://assistant.google.com
9. "Microsoft Cortana." [Online]. Available: https://www.microsoft.com/en-us/cortana
10. A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," mar 2013. [Online]. Available: https://arxiv.org/abs/1303.5778
11. A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," dec 2014. [Online]. Available: http://arxiv.org/abs/1412.5567
12. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," sep 2016. [Online]. Available: https://arxiv.org/abs/1609.03499
13. The BBC, "AI image recognition fooled by single pixel change," 2016. [Online]. Available: https://www.bbc.com/news/technology-41845878
14. The New York Times, "Alexa and Siri Can Hear This Hidden Command. You Can't," 2016. [Online]. Available: https://www.nytimes.com/2018/05/10/technology/alexa-siri-hidden-command-audio-attacks.html
15. Google Accident, "A Google self-driving car caused a crash for the first time," 2016. [Online]. Available: https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report
16. Uber Accident, "After Fatal Uber Crash, a Self-Driving Start-Up Moves Forward," 2018. [Online]. Available: https://www.nytimes.com/2018/05/07/technology/uber-crash-autonomous-driveai.html
17. A. Karpathy, "Software 2.0," https://medium.com/@karpathy/software-2-0-a64152b37c35, 2018.
18. K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

19. L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," *The 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2018)*, 2018.
20. J. Kim, R. Feldt, and S. Yoo, "Guiding Deep Learning System Testing using Surprise Adequacy," *ArXiv e-prints*, Aug. 2018.
21. T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Systematic Testing of Convolutional Neural Networks for Autonomous Driving," in *Reliable Machine Learning in the Wild*, 2017.
22. M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018, 2018.
23. Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 303–314.
24. G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," *arXiv preprint arXiv:1711.09576*, 2017.
25. N. Carlini and D. Wagner, "Audio Adversarial Examples: Targeted Attacks on Speech-to-Text," jan 2018. [Online]. Available: http://arxiv.org/abs/1801.01944
26. X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter, "CommanderSong: A Systematic Approach for Practical Adversarial Voice Recognition," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, pp. 49–64. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/yuan-xuejing
27. C. Kereliuk, B. L. Sturm, and J. Larsen, "Deep Learning and Music Adversaries," *ArXiv e-prints*, Jul. 2015.
28. Y. Gong and C. Poellabauer, "Crafting Adversarial Examples For Speech Paralinguistics Applications," *ArXiv e-prints*, Nov. 2017.
29. S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.
30. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
31. K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.
32. A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.
33. Y. Miao, M. Gowayyed, and F. Metze, "Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding," in *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 2015, pp. 167–174.
34. I. Jolliffe, "Principal Component Analysis," in *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 1094–1096.
35. M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.
36. C. W. Omlin and C. L. Giles, "Constructing Deterministic Finite-State Automata in Recurrent Neural Networks," *Journal of the ACM (JACM)*, vol. 43, no. 6, pp. 937–972, 1996.
37. P. Rastogi, R. Cotterell, and J. Eisner, "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*, 2016, pp. 623–633.

38. A. Gill, *Introduction to the Theory of Finite-State Machines*, ser. McGraw-Hill electronic sciences series. McGraw-Hill, 1962. [Online]. Available: https://books.google.com.sg/books?id=2IzQAAAAMAAJ
39. B. G. Horne, C. L. Giles, P. C. Collingwood, S. O. Computing, M. Sci, P. Tino, and P. Tino, "Finite State Machines and Recurrent Neural Networks – Automata and Dynamical Systems Approaches," in *Neural Networks and Pattern Recognition*. Academic Press, 1998, pp. 171–220.
40. J. F. Thompson, B. K. Soni, and N. P. Weatherill, *Handbook of Grid Generation*. CRC press, 1998.
41. T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep., 1998.
42. "Mozilla's DeepSpeech," https://github.com/mozilla/DeepSpeech, 2018.
43. C. Cieri, D. Miller, and K. Walker, "The fisher corpus: a resource for the next generations of speech-to-text." in *LREC*, vol. 4, 2004, pp. 69–71.
44. V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5206–5210.
45. J. J. Godfrey, E. C. Holliman, and J. McDaniel, "Switchboard: Telephone speech corpus for research and development," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 1. IEEE, 1992, pp. 517–520.
46. "Commom Voice Dataset." [Online]. Available: https://voice.mozilla.org/en/datasets
47. Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.
48. L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang, "Combinatorial Testing for Deep Learning Systems," *arXiv e-prints*, p. arXiv:1806.07723, Jun. 2018.
49. Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic Testing for Deep Neural Networks," 2018.
50. A. Odena and I. Goodfellow, "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing," 2018.
51. X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Coverage-Guided Fuzzing for Deep Neural Networks," sep 2018. [Online]. Available: http://arxiv.org/abs/1809.01266
52. L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, Oct 2018, pp. 100–111.
53. C. W. Omlin and C. L. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural networks*, vol. 9, no. 1, pp. 41–52, 1996.
54. A. L. Cechin, D. R. P. Simon, and K. Stertz, "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*, 2003, p. 73.
55. Q. Wang, K. Zhang, A. G. Ororbia, II, X. Xing, X. Liu, and C. L. Giles, "An empirical evaluation of rule extraction from recurrent neural networks," *Neural Comput.*, vol. 30, no. 9, pp. 2568–2591, Sep. 2018. [Online]. Available: https://doi-org.ezlibproxy1.ntu.edu.sg/10.1162/neco_a_01111
56. B.-J. Hou and Z.-H. Zhou, "Learning with Interpretable Structure from RNN," oct 2018. [Online]. Available: http://arxiv.org/abs/1810.10708
57. L. Ma, F. Juefei-Xu, M. Xue, Q. Hu, S. Chen, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Secure Deep Learning Engineering: A Software Quality Assurance Perspective," *arXiv e-prints*, p. arXiv:1810.04538, Oct. 2018.