9-2020

Marble: Model-based robustness analysis of stateful deep learning systems

Xiaoning DU

Yi LI

Xiaofei XIE
Singapore Management University, xfxie@smu.edu.sg

Lei MA

Yang LIU

See next page for additional authors

MARBLE: Model-based Robustness Analysis of Stateful Deep Learning Systems

Xiaoning Du
Nanyang Technological University
Singapore

Yi Li
Nanyang Technological University
Singapore

Xiaofei Xie*
Nanyang Technological University
Singapore

Lei Ma
Kyushu University
Japan

Yang Liu
Nanyang Technological University
Singapore

Jianjun Zhao
Kyushu University
Japan

ABSTRACT

State-of-the-art deep learning (DL) systems are vulnerable to adversarial examples, which hinders their potential adoption in safety- and security-critical scenarios. While some recent progress has been made in analyzing the robustness of feed-forward neural networks, the robustness analysis for stateful DL systems, such as recurrent neural networks (RNNs), still remains largely uncharted. In this paper, we propose MARBLE, a model-based approach for quantitative robustness analysis of real-world RNN-based DL systems. MARBLE builds a probabilistic model to compactly characterize the robustness of RNNs through abstraction. Furthermore, we propose an iterative refinement algorithm to derive a precise abstraction, which enables accurate quantification of the robustness measurement. We evaluate the effectiveness of MARBLE on both LSTM and GRU models trained separately with three popular natural language datasets. The results demonstrate that (1) our refinement algorithm is more efficient in deriving an accurate abstraction than the random strategy, and (2) MARBLE enables quantitative robustness analysis, in rendering better efficiency, accuracy, and scalability than the state-of-the-art techniques.

ACM Reference Format:
Xiaoning Du, Yi Li, Xiaofei Xie, Lei Ma, Yang Liu, and Jianjun Zhao. 2020. MARBLE: Model-based Robustness Analysis of Stateful Deep Learning Systems. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3324884.3416564

1 INTRODUCTION

With the booming of big data and hardware acceleration, deep learning (DL) has achieved tremendous success in many applications such as image processing [14], speech recognition [3, 29], video and board games [41, 47]. In spite of achieving high accuracy, deep neural networks (DNNs) are still vulnerable to adversarial attacks [8, 11, 12]. For example, an image classifier can easily be fooled by pixel-level perturbations [52] or inevitable noise in physical-world situations [22]. Hence, quality and reliability assurance of DL systems are urgently needed, especially for those applied in safety- and security-critical scenarios.

Robustness analysis aims to estimate the capability of a neural network in tolerating input perturbations, which naturally occur in the physical environment or are intentionally applied by malicious parties (*e.g.*, adversarial attacks). A neural network is said to be robust, if its prediction results could not be misled by any small perturbations (*e.g.*, imperceptible by human). In fact, it has been shown that most neural networks are vulnerable to small manipulations to the inputs [8, 51]. By far, there are two types of analysis methods to quantify robustness: *robustness verification* [6, 24] that provides theoretical guarantees on the level of perturbations a network is immune to, and *robustness quantification* [10, 38] which estimates the robustness score of a neural network with the difficulty of finding inputs that lead to incorrect prediction results. Intuitively, the harder it is to generate such inputs, the more robust the model is.

Moreover, the quantification methods focus on either generating human imperceptible adversarial input perturbations [27] to manipulate a DNN's decision (*e.g.*, gradient-based attacks), or producing systematic tests [38, 43] to simulate realistic noise and transformations that might occur in physical environment. Generally speaking, the quantification methods are often more scalable than the verification-based techniques, and they can be generalized to different network architectures more easily. However, it is non-trivial to improve model robustness, even with samples explicitly exposing the weaknesses. Lacking robustness over an input sample indicates that the result produced by the DNN is potentially unreliable. If the robustness analysis can be performed in real-time, a decision-making system is able to fallback to alternative backup plans whenever the DNN reveals poor robustness.

Existing methods face challenges in terms of efficiency and applicability, especially in handling large models. In particular, the testing- and attack-based techniques often require a large number of inputs to be generated and tested, which is expensive and hardly applicable to real-time applications. In addition, the vast majority of existing robustness analysis techniques only focus on feed-forward neural networks (FNNs), leaving other types of networks, such as recurrent neural networks (RNNs), largely untouched. RNNs are designed to process sequential inputs (*e.g.*, natural languages, audios,

*Xiaofei Xie (xfxie@ntu.edu.sg) is the corresponding author.

ASE '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3416564

and signals), and also vulnerable to adversarial attacks. For instance, the RNN-based toxic contents (*e.g.*, insults and violence) detector in online discussions can be circumvented by replacing or modifying a single word [35]. To our best knowledge, POPQORN [33] is currently the only work addressing the robustness verification of RNNs. It assumes continuous input domains and relies on expensive bound calculations propagated through layers (*i.e.*, on unrolled RNNs). As a result, it is rather imprecise when handling discrete domain inputs, such as natural language texts, and faces scalability and efficiency issues. The problem of efficiently evaluating the robustness of RNNs in processing sequential inputs remains an open challenge. Solving this problem is an essential step towards improving the reliability of real-time applications, such as machine translation for simultaneous interpretation, speech recognition, and perception tasks in cyber-physical systems.

To address the aftermentioned issues, we propose a model-based approach, named Marble, for evaluating the robustness of RNNs effectively. The aim is to build an abstract model offline and enable light-weight robustness estimation online for real-time applications. To overcome the performance challenges and avoid making many perturbations at runtime, we construct a robustness-aware abstract model for RNNs to capture the behaviors of an RNN given various inputs in a compact form. Due to the stateful nature of RNNs, the output produced at each step is affected by both the input element and the context of it. For example, in sentiment analysis, the positive/negative opinions predicted at a certain word within a sentence are affected by the word itself and also words before it. Our abstract model maintains previously seen states and transitions among states, which are essential in characterizing how different inputs are handled under various contexts.

The contextual information can also be useful in estimating robustness. Intuitively, under a similar context, the robustness over the same input element is expected to be similar as well. For instance, when processing two movie reviews, "I love this movie" and "I love that movie", an RNN often maintains very similar contexts before processing the final word "movie". With this insight, we compute robustness measures for training inputs and encode this information into the abstract model, which is then used to estimate the robustness of new (previously unseen) test inputs. This makes real-time identification of unreliable predictions possible and could improve the overall quality of the decision making process.

In particular, we build an abstract model based on Markov Decision Process (MDP) for a given RNN model, and propose a refinement algorithm to iteratively improve the precision of the abstraction in quantifying robustness. The refinement process continues until the robustness estimation error (*i.e.*, the difference of the robustness estimation from MDP and the dynamic input perturbations) is reduced under a threshold. We applied Marble to quantify the robustness of six RNNs which are trained for news title classification, toxic comment detection and sentiment analysis. The results show that Marble is more effective in refining the abstract model than random strategies and produces (at least 2 times) smaller accurate MDPs. Marble is more efficient than the state-of-the-art technique – the average time taken by Marble per input is 0.03 seconds while POPQORN requires 52.8 minutes. Marble is also more precise – it achieves higher attack success rate on the least robust inputs and lower attack success rate on the most robust inputs, compared with POPQORN.

The main contributions of this paper are three-fold.

- We proposed to abstract RNN as an MDP model in order to perform a quantitative robustness analysis of an RNN.
- We introduced a mutation-based refinement technique to continuously refine the MDP for more accurate robustness estimation of an RNN.
- We performed an in-depth evaluation on two popular datasets, to demonstrate the effectiveness of the refinement algorithm and the estimation precision of Marble.

2 PROBLEM FORMULATION

2.1 Formalization of RNN and Traces

Definition 2.1 (Recurrent Neural Network (RNN)). An RNN is a tuple $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{O}, f)$ such that $\mathcal{X}$, $\mathcal{S}$, and $\mathcal{O}$ denote the domains of the inputs, hidden states, and outputs, respectively, and f is a differentiable parameterized function. $\mathcal{R}$ takes as input a sequence $\mathbf{x} \in \mathcal{X}^n$ of length n, maintains a sequence of hidden states $\mathbf{s} \in \mathcal{S}^{n+1}$ of length $n+1$, and applies the function f on each state and input pair (s_i, x_i) to produce an output sequence $\mathbf{y} \in \mathcal{O}^n$ such that $(s_{i+1}, y_i) = f(s_i, x_i)$, where x_i, s_i, and y_i denote the i-th element of the respective sequences.

Following [46], we formalize an RNN abstractly as above. The hidden states $\mathcal{S} \subseteq \mathbb{R}^m$ are m-dimensional real number vectors and the initial state s_0 is a vector of m zeros. s_i^d is used to denote the d-th dimension of the state vector s_i, where $d \in [0, m)$.

Given an RNN, each input sequence $\mathbf{x}$ induces a finite sequence of state transitions, which forms a *trace* denoted by $\mathbf{t}(\mathbf{x})$. The i-th element of a trace, denoted by $t_i(\mathbf{x})$, is the transition from s_i to s_{i+1} after accepting an input element x_i and producing an output y_i. The traces of an RNN can be represented compactly as a Finite State Transducer (FST) [25], which is formally defined as follows.

Definition 2.2 (Finite State Transducer (FST)). Given an RNN $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{O}, f)$, we define its traces induced by a set of inputs $D \in 2^{\mathcal{X}^n}$ as a finite state transducer, $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$, where $\mathcal{S}$ is a non-empty finite set of states, $\mathcal{X}$ is the input alphabet, $\mathcal{O}$ is the output alphabet, $s_0 \in \mathcal{S}$ is the initial state, $F \subseteq \mathcal{S}$ is the set of final states, and $\delta \subseteq \mathcal{S} \times \mathcal{X} \times \mathcal{O} \times \mathcal{S}$ is the transition relation.

Example 2.3. Fig. 1 gives an example of how to represent the RNN behaviors as an FST, with transitions represented by solid black arrows. The RNN, $\mathcal{R}$, is assumed to work on sentiment analysis and is trained to classify movie reviews into two categories, namely, negative and positive. Note that 0 is used to represent negative opinions, and 1 is used for positive ones.

Given a set of movie reviews, $D = \{$["*i*", "*really*", "*like*", "*this*", "*movie*"], ["*i*", "*like*", "*this*", "*movie*"], ["*we*", "*like*", "*this*", "*movie*"]$\}$, three traces are induced. They share the same initial state s_0 and consist of five, four and four transitions, respectively. Transitions are represented by arrows, originating from one state and transit to another state, labeling with its input and output pair. For the transition from s_0 to s_1, the label "$i : 0$" denotes that this transition is induced by the input element i and emits the output 0.

For the FST defined over these three traces, the set of input alphabet is $\mathcal{X} = \{$"*i*", "*we*", "*really*", "*like*", "*the*", "*that*", "*this*", "*movie*"$\}$,

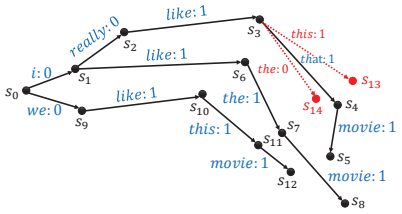


Figure 1: FST Example.

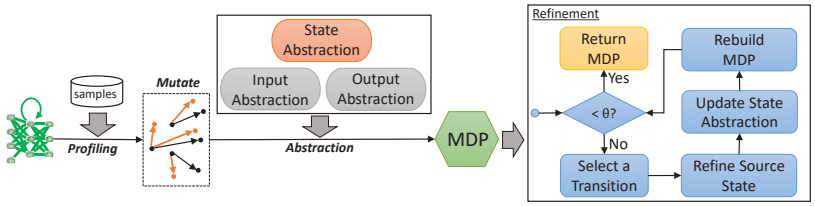


Figure 2: The overview of MARBLE.

the set of states is $\mathcal{S} = \{s_0, \ldots, s_{12}\}$, the set of outputs is $O = \{0, 1\}$, the initial state is s_0, the set of final states is $F = \{s_5, s_8, s_{12}\}$, and the transition relation δ defines the set of 12 transitions, *e.g.*, $(s_0, \text{“}i\text{”}, 0, s_1)$, and $(s_1, \text{“}really\text{”}, 0, s_2)$.

2.2 Pointwise Robustness of an RNN

The quantification of FNN robustness has been widely studied in the literature [9, 54], whereas little work has been done to address the robustness of RNN models. Unlike FNN that takes an input as a whole, RNN consumes a sequential input element-by-element, and perturbations on each input element may cause different levels of impact to the model outputs [18, 30]. Theoretically, the change at a solitary input frame could make a long-lasting influence on the process of upcoming frames. Also due to the complicated training and working mechanism of RNNs, even well-trained RNNs tend to reveal poor robustness to perturbations at a single input frame. In line with the assumptions in POPQORN, we study the pointwise robustness of RNNs, instead of allowing mutations over a wide range of input frames. Such a strict constraint can be more instructive for the quality assurance of RNNs at current stage.

To measure the robustness of an RNN model w.r.t. an input, we allow the input to be perturbed within a certain range, and observe their impact on the induced traces. Instead of searching for the lower bound of perturbations that would alter the RNN decision, we look into how likely perturbations within the range could make a difference. Because the lowerbound measure over discrete data, *e.g.*, natural language, is highly influenced by the sparsity of the data. To precisely quantify the pointwise robustness, we first specify how input elements are to be perturbed, formally defined by a *pointwise mutation function*. Note that, for a given set Q, the probability distribution over Q is a function $d : Q \mapsto [0, 1]$ such that $\Sigma_{q \in Q} d(q) = 1$. We denote the set of all probability distributions over Q as $\text{Dist}(Q)$.

Definition 2.4 (Pointwise Mutation Function). A pointwise mutation function $\mu \subseteq \mathcal{X} \times \text{Dist}(\mathcal{X})$ maps an input element $x \in \mathcal{X}$ to a probability distribution of the possible mutants $x' \in \mathcal{X}$, such that $\mu(x, x')$ is the probability of mutating x into x'.

The pointwise mutation function can be used to describe various types of perturbations to the inputs. For example, p-norm ball [7, 33] is often used in the literature to describe perturbations happened uniformly within a sphere around the given input. In general, the robustness of a model can be examined against different real-world attack patterns by simulating the ranges and frequencies of the potential mutations appeared in practice.

We define the *0-step pointwise robustness* of an RNN at an input element x_i as the likelihood that the immediate next transition t_i induced by x_i stays unchanged after some *perturbation* is applied to x_i. This definition can be further generalized to observe the impact of the perturbation only after a certain number of transitions, instead of the immediate next one.

Definition 2.5 (k-Step Pointwise Robustness). Let $\mathcal{R}$ be an RNN and $\mathbf{x} \in \mathcal{X}^n$ be an input of length n, respectively. Let x_i be the input element at the i-th position of $\mathbf{x}$, where $i < n$. Let μ be a pointwise mutation function generating a new input $\mathbf{x}[x_i \to x_i']$ by mutating x_i. The k-step pointwise robustness of $\mathcal{R}$ w.r.t. $\mathbf{x}$ at the i-th position against perturbations defined by μ, denoted by $\gamma^k(\mathcal{R}, \mathbf{x}, i, \mu)$, is given as the probability that the trace induced by $\mathbf{x}'$ produces the same output at the $(i + k)$-th transition, where $(i + k) \leq n$. More formally,

$$\gamma^k(\mathcal{R}, \mathbf{x}, i, \mu) = \sum_{x_i' \in \mathcal{X} \cdot t_{i+k}(\mathbf{x}) \stackrel{o}{=} t_{i+k}(\mathbf{x}[x_i \to x_i'])} \mu(x_i, x_i'),$$

where $\stackrel{o}{=}$ denotes the equality of outputs for both sides.

The pointwise robustness definition is parameterized by the number of steps, after which the impact of the perturbation to the model output are observed. Specifically, $\mathbf{t}(\mathbf{x})$ and $\mathbf{t}(\mathbf{x}[x_i \to x_i'])$ represent the two traces induced by $\mathbf{x}$, before and after x_i is mutated, respectively. The k-step pointwise robustness only observes the outputs produced at the $(i+k)$-th transition, denoted by $t_{i+k}(\mathbf{x})$, and ignores any intermediate states and transitions. In particular, for the 0-step pointwise robustness, we observe how the transition t_i induced by x_i is affected by perturbations over x_i. Instead of parameterizing pointwise robustness by the input sequence and position indices, we sometimes use an equivalent notation $\gamma^0(\mathcal{R}, s_i, x_i, \mu)$ such that,

$$\gamma^0(\mathcal{R}, s_i, x_i, \mu) = \sum_{x_i' \in \mathcal{X} \cdot f(s_i, x_i) \stackrel{o}{=} f(x_i, x_i')} \mu(x_i, x_i').$$

Example 2.6. Following Fig. 1, we briefly illustrate how the 0-step pointwise robustness is calculated with a simple example. Here we compute the robustness of $\mathcal{R}$ over the input sequence [“*i*”, “*really*”, “*like*”, “*that*”, “*movie*”] at its fourth input element, *i.e.*, “*that*”. For the mutation function μ, we assume that “*that*” can be mutated into any element in {“*the*”, “*that*”, “*this*”} with an equal probability, *i.e.*, $\frac{1}{3}$. The perturbation of “*that*” would affect the transition right after the state s_3, and we assume the two new transitions induced by the replacements are $(s_3, \text{“}this\text{”}, 1, s_{13})$, $(s_3, \text{“}the\text{”}, 0, s_{14})$, respectively. For the replacement to itself, the transition keeps still as $(s_3, \text{“}that\text{”}, 1, s_4)$. Obviously, among all these three transitions, two out of them emit the same output 1 as originally. Therefore, the 0-step robustness of $\mathcal{R}$ over the input sequence [“*i*”, “*really*”, “*like*”, “*that*”, “*movie*”] at “*that*” is $\frac{2}{3}$.

2.3 Overview

Following Definition 2.5, the pointwise robustness of an RNN can be estimated through a random sampling from mutations. One can randomly generate mutants according to the probabilistic distribution given by the mutation function, and observe their impact on the model outputs directly. Yet, this sampling approach likely does not scale in practice. To get precise robustness measurements w.r.t. even a single input, the number of mutants required can be huge if the RNN model is non-trivial.

To overcome this challenge, we propose a model-based analysis technique, Marble, to efficiently compute the pointwise robustness of a given RNN with respect to an input element. Figure 2 summarizes the workflow of Marble. We first construct a robustness-aware abstract model by observing how the RNN's behaviors are affected by pointwise mutations. Specifically, Marble profiles the RNN's behaviors on the training data and constructs an FST. Then, for each transition within the FST, Marble examines its (0-step) robustness with input replacements sampled following the mutation probabilistic distribution. The mutation function is specific to the application domains (*e.g.*, audio and image) and pre-defined by the users. We then build an abstract model, in the form of a Markov Decision Process (MDP), by abstracting the states and transitions in the FST, and annotate the MDP with robustness measurements adapted with the abstraction.

Given an input, we can then compute a robustness score, regarding each input frame, based on the MDP model. If the obtained score is inconsistent with the result of the dynamic *mutation sampling* (*i.e.*, their difference exceeds a threshold), we iteratively refine the abstract model so that the robustness can be more precisely quantified. Finally, we obtain a precise abstract model after the results of all training data from the abstract model is consistent with the *mutation sampling*. Based on the abstract model, we can analyze the robustness of the RNN against new inputs that follow a similar distribution of training data efficiently.

3 RNN ABSTRACTION USING LABELED MDP

As shown in Definition 2.5, the robustness of an RNN is defined with respect to a concrete input sequence at a certain position. Yet, the number of states and traces enabled during the training stage of an RNN can be huge. It can be impractical, during runtime, to compute the robustness for each trace individually, with either classical mathematical estimation methods [33] or mutation testing techniques [23]. To make the computation more efficient, we now introduce an RNN abstraction model based on labeled MDPs.

3.1 Aggregated Pointwise Robustness

The key idea in deriving a suitable abstraction is to group the traces with similar robustness measures together, which can be used to estimate the robustness of an RNN for various inputs at various contexts in a more cost-effective manner. Given a set of states $\tilde{S}$, from which transitions are induced by an input frame x, we can calculate the 0-step robustness for each such transition, against its respective output. We define the *aggregated pointwise robustness* at $\tilde{S}$ to reflect the overall robustness of a set of input-induced traces against perturbations. The collection of transitions from $\tilde{S}$ may emit different outputs. To ease the computation of the probability of keeping the original outputs collectively, we record the probabilities emitting different outputs as a distribution over the output domain. Specifically, we denote the probability of emitting y at $s \in \tilde{S}$ when mutating input x with μ as:

$$\gamma^0(\mathcal{R}, s, x, \mu)[y] = \sum_{x' \in \mathcal{X} \cdot f(s,x') \overset{o}{=} (\_, y)} \mu(x, x').$$

We use the notation $\gamma^0(\mathcal{R}, s, x, \mu)$ to denote the probability distribution over the output domain, and omit the parameters $\mathcal{R}$ and μ when they are clear from the context. Given that it is a categorical distribution [4], the *mean* of this distribution is a vector $\mathbf{p}$, of length $|O|$, where $p_i = \gamma^0(\mathcal{R}, s, x, \mu)[y_i]$ for any $y_i \in O$. Further, $\gamma^0(\mathcal{R}, s, x, \mu)$ is sometimes also abused to represent the *mean* vector. For an instance originally yielding y, the robustness is obtained via taking the corresponding probability $\gamma^0(\mathcal{R}, s, x, \mu)[y]$. Now, the aggregated robustness definition requires integrating a set of distributions observed at different states. A common practice to reconcile different probability distributions is to take the average of their probability densities [28]. Hence, we define the aggregated pointwise robustness as follows.

Definition 3.1 (Aggregated Pointwise Robustness). Let $\mathcal{R}$ be an RNN and $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ be its traces induced by a set of inputs D. Let x be an input element and μ be a pointwise mutation function. Let $\tilde{S} \subseteq \mathcal{S}$ be a set of states such that for all $s \in \tilde{S}$, $f(s, x)$ emits the same output. The aggregated pointwise robustness of $\mathcal{R}$ at $\tilde{S}$ w.r.t. x against perturbations defined by μ, denoted by $\Gamma(\mathcal{R}, \tilde{S}, x, \mu)$, is given as the average of the 0-step pointwise robustness of $\mathcal{R}$ w.r.t. the states in $\tilde{S}$. More formally,

$$\Gamma(\mathcal{R}, \tilde{S}, x, \mu) = \frac{1}{|\tilde{S}|} \sum_{s \in \tilde{S}} \gamma^0(\mathcal{R}, s, x, \mu). \tag{1}$$

The aggregated pointwise robustness is essentially the average value of all pointwise robustness observations for a set of states. In particular, for an output y, we have,

$$\Gamma(\mathcal{R}, \tilde{S}, x, \mu)[y] = \frac{1}{|\tilde{S}|} \sum_{s \in \tilde{S}} \left(\gamma^0(\mathcal{R}, s, x, \mu)[y] \right),$$

which takes the average of the probabilities of emitting y in different distributions. With Eq. (1), we are able to represent the overall robustness at a set of states obeying proper statistical rules.

3.2 Abstraction as Labeled MDP

Abstraction on states is essentially dividing the state space such that states with similar robustness measures are clustered. After such clustering, an abstract state may have multiple non-deterministic outgoing transitions even after receiving the same input. Markov Decision Process (MDP) [45] is widely used to describe probabilistic systems, and to characterize the probabilistic state transitions triggered by different input elements. Here, we use the labeled MDP to formalize the abstraction of RNN traces, and the detailed definition is given in Definition 3.2. The labeling function is used to record the robustness measure of each state and input element pair.

Definition 3.2 (Labeled Markov Decision Process). A labeled MDP, $\mathcal{M}$, is a tuple $(\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{O}, \hat{s}_0, \hat{\delta}, \eta)$ consisting of a finite set $\hat{\mathcal{S}}$ of states, a finite set $\hat{\mathcal{X}}$ of input alphabet, an initial state $\hat{s}_0$, a finite probabilistic transition relation $\hat{\delta} \subseteq \hat{\mathcal{S}} \times \hat{\mathcal{X}} \times \text{Dist}(\hat{\mathcal{S}})$, where states and inputs are in relation with distributions of successor states, and a labeling

function $\eta \subseteq \hat{\mathcal{S}} \times \hat{\mathcal{X}} \times \text{Dist}(\mathcal{O})$ which maps the state and input pairs to distributions of outputs.

Now we demonstrate how to derive a labeled MDP from a set of observed traces represented as an FST. In order to represent the behaviors of an RNN in a more compact manner, we allow abstractions over the state domain, as well as the input and output domains, and more details about the abstraction functions will be discussed in Section 4.1. Given an FST $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ of an RNN $\mathcal{R}$ on the input set D, an input abstraction function $\lambda_I : \mathcal{X} \mapsto \hat{\mathcal{X}}$, a state abstraction function $\lambda_S : \mathcal{S} \mapsto \hat{\mathcal{S}}$ and an output abstraction function $\lambda_O : \mathcal{O} \mapsto \hat{\mathcal{O}}$, we can establish a labeled MDP $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta)$ for $\mathcal{T}_{\mathcal{R}}(D)$. The input, state and output domains in $\mathcal{M}$ are abstracted from those in $\mathcal{T}_{\mathcal{R}}(D)$ with the abstraction functions. Specifically, $\hat{s}_0 = \lambda_S(s_0)$, and for each transition $(s, x, s', y) \in \delta$, it is abstracted as $(\hat{s} = \lambda_S(s), \hat{x} = \lambda_I(x), \hat{s}' = \lambda_S(s'), \hat{y} = \lambda_O(y))$ and included as an abstract transition in $\mathcal{M}$. With these abstractions, the probabilistic transition relation $\hat{\delta}$ and the labeling function η can be derived accordingly. $\hat{\delta}(\hat{s}, \hat{x}, \hat{s}')$ denotes the conditional probability of visiting $\hat{s}'$ given the current abstract state $\hat{s}$ and an abstract input element $\hat{x}$, and $\Sigma_{\hat{s}' \in \hat{\mathcal{S}}} \hat{\delta}(\hat{s}, \hat{x}, \hat{s}') = 1$. The transition probability is calculated as the number of concrete transitions from $\hat{s}$ to $\hat{s}'$ via $\hat{x}$ over the number of all outgoing concrete transitions from $\hat{s}$ accepting abstract input element $\hat{x}$, *i.e.*,

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

The labeling function η records the 0-step aggregated pointwise robustness of $\mathcal{R}$ under a mutation function μ. Note that the robustness definition in Definition 3.1 is presented under the concrete input and output domains, and here we adjust it to reflect their abstract counterparts. Similar to Definition 3.1, given an abstract state $\hat{s}$ and an abstract input $\hat{x}$, the outputs of the corresponding concrete state and input pairs can vary. Therefore, we also label the robustness under abstract states and abstract inputs as a distribution over the output domain. Formally, we define 0-step aggregated pointwise robustness at $\hat{s}$ over $\hat{x}$, when emitting the abstract output $\hat{y}$, as,

$$\eta(\hat{s}, \hat{x}, \hat{y}) = \frac{1}{|S_{\hat{x}}|} \sum_{(s,x) \in S_{\hat{x}}} \left(\sum_{y \in \hat{y}} \gamma^0(\mathcal{R}, s, x, \mu)[y] \right), \tag{2}$$

where $S_{\hat{x}} = \{(s, x) \mid \exists (s, x, \_, \_) \in \delta \text{ for } s \in \hat{s}, x \in \hat{x}\}$ is the set of state-and-input pairs which source from concrete states in $\hat{s}$ and accept concrete input elements in $\hat{x}$. The inner sum calculates the robustness at s over x, while relaxing the constraint on the expected output to be within the abstract output $\hat{y}$. Then, we take the average value for robustness observations over all pairs inside $S_{\hat{x}}$.

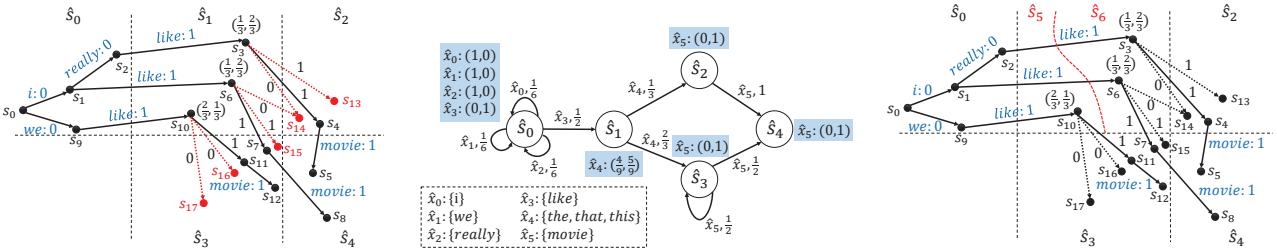
Example 3.3. Continuing with the example in Fig. 1, we draw the abstract MDP model derived from it in Fig. 3(a) and Fig. 3(b). States and transitions induced by mutations over "the", "that", and "this" are highlighted as red dots and red dashed arrows, respectively. We assume that these three words share the same mutation probability distribution, *i.e.*, each could be mutated to others (including itself) with equal probability, *i.e.*, $\frac{1}{3}$. The corresponding outputs, either 0 or 1, are annotated along the red arrows. For instance, both transitions from s_{10} to s_{16} and from s_{10} to s_{17} emit output 0, and they are induced by replacing "this" with "the" and "that", respectively. Input elements are omitted to keep the illustration concise. Fig. 3(b) shows the MDP abstraction of the concrete traces in Fig. 3(a). Now we elaborate on the *input/output abstraction* and the *state/transition abstraction* carried out during the MDP construction.

In this example, we assume an identity abstraction function for the output space. The full input abstraction function is presented in the dashed box at Fig. 3(a). For instance, "the", "that" and "this" are abstracted as $\hat{x}_4$. The state space are abstracted via the grids drawn in dashed lines in Fig. 3(a), resulting in five abstract states, *i.e.*, $\hat{s}_0$, $\hat{s}_1$, $\hat{s}_2$, $\hat{s}_3$, and $\hat{s}_4$, each of which is mapped to a set of concrete states inside the corresponding grid. For each abstract state $\hat{s}$, we can then compute the transition function by deciding the set of abstract inputs accepted at $\hat{s}$ and the probabilistic distributions of the successor states under each abstract input. For example, there are three concrete transitions originated from $\hat{s}_1$, which lead to two abstract successor states: $\hat{s}_2$ and $\hat{s}_3$. The abstract state $\hat{s}_1$ accepts only one abstract input, $\hat{x}_4$, inducing a distribution of the successor states, *i.e.*, $\{\hat{\delta}(\hat{s}_1, \hat{x}_4, \hat{s}_2) = \frac{1}{3}, \hat{\delta}(\hat{s}_1, \hat{x}_4, \hat{s}_3) = \frac{2}{3}\}$. The transition probabilities are marked over the transitions between abstract states in Fig. 3(b) as abstract input and probability pairs.

Finally, we show how the labeling function is computed, which signifies the 0-step aggregated pointwise robustness. In Fig. 3(b), this part of information is highlighted with blue boxes aside each abstract state. According to Eq. (2), the 0-step aggregated pointwise robustness is calculated by taking the average of robustness observed concretely. We take the abstract state $\hat{s}_1$ and the abstract input $\hat{x}_4$ as example. From the concrete state transitions in Fig. 3(a), we can see that there are three concrete states within $\hat{s}_1$, and all of them accept concrete input elements in $\hat{x}_4$. Specifically, at s_3, according to the mutation function, the probability to yield output 0 is $\frac{1}{3}$, and to yield 1 is $\frac{2}{3}$. The probability distribution over the outputs are marked for s_3, s_6 and s_{10}, when accepting input elements belong to $\hat{x}_4$. Finally, we can calculate the 0-step aggregated pointwise robustness at $\hat{s}_1$ over $\hat{x}_4$, *i.e.*, $\eta(\hat{s}_1, \hat{x}_4, 0) = (\frac{1}{3} + \frac{1}{3} + \frac{2}{3})/3 = \frac{4}{9}$ and $\eta(\hat{s}_1, \hat{x}_4, 1) = (\frac{2}{3} + \frac{2}{3} + \frac{1}{3})/3 = \frac{5}{9}$. Hence, we label the output probability distribution of $\hat{x}_4$ at $\hat{s}_1$ as $(\frac{4}{9}, \frac{5}{9})$. The concrete distribution annotations are omitted in Fig. 3(a) for other concrete states and transitions, since they can be easily computed when no mutations are applied to them. For the concrete transition $(s_0, \text{"}i\text{"}, 0, s_1)$, the output probability distribution is $(1, 0)$, indicating output 0 is yielded with probability 1. Since there is only one concrete state in $\hat{s}_0$ accepting $\hat{x}_0$, the aggregated robustness is directly $(1, 0)$. Similarly, we can calculate the output distribution for each abstract state over its accepting abstract inputs.

3.3 k-step Aggregated Pointwise Robustness

Based on the abstracted MDP model, we define the k-step aggregated pointwise robustness. Firstly, we define traces over the MDP induced by an abstract input sequence starting from a designated abstract state. Given an MDP $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}, \hat{\delta}, \eta)$, a start abstract state $\hat{s}_0 \in \hat{\mathcal{S}}$ and an abstract input $\hat{\mathbf{x}} \in \hat{\mathcal{X}}^n$ of length n, we denote the set of traces over $\mathcal{M}$ triggered by $\hat{\mathbf{x}}$ and starting from $\hat{s}_0$ as $\Pi(\hat{\mathbf{x}}, \hat{s}_0)$. The i-th element in a trace $\pi \in \Pi(\hat{\mathbf{x}}, \hat{s}_0)$ is the transition from $\hat{s}_i$ to $\hat{s}_{i+1}$ via $\hat{x}_i$. In particular, the 0-th transition is triggered by $\hat{x}_0$ and transits from $\hat{s}_0$ to $\hat{s}_1$. Moreover, we use $\rho(\pi)$ to denote the trace probability of π, which is the product of transit probabilities for



(a) Abstraction over the state space. (b) MDP abstraction and modeling. (c) Refinement of the State Abstraction.

Figure 3: MDP construction and refinement.

transitions in π, formally, $\rho(\pi) = \prod_{i=0}^{n-1} \hat{\delta}(\hat{s}_i, \hat{x}_i, \hat{s}_{i+1})$. Specially, we define ϵ as the empty abstract input element, and for any abstract state, the self-transition via ϵ happens with probability 1. Hence, given ϵ as the input sequence of length 1, we have $\Pi(\epsilon, \hat{s}_0)$ contains the *only* trace with a single self-transition of $\hat{s}_0$. To facilitate the notations, we use $\mathbf{x}[i\colon j]$ to denote the subsequence of a sequence $\mathbf{x}$, which starts from its i-th element and ends at the (j-1)-th element; if $i \geq j$, the subsequence is ϵ. Also, we use π_r to denote the reached state of its last transition. We give the definition of k-step aggregated pointwise robustness in Definition 3.4.

Definition 3.4 (k-step Aggregated Pointwise Robustness). Given an RNN $\mathcal{R}$, let $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{O}, \hat{s}_0, \hat{\delta}, \eta)$ be a labeled MDP established for it. Let $\mathbf{x} \in \mathcal{X}^n$ be an input of length n, and the corresponding abstract input is $\hat{\mathbf{x}}$ with $\hat{x}_i = \lambda_I(x_i)$ for $i < n$. The k-step aggregated pointwise robustness of $\mathcal{R}$ w.r.t. $\mathbf{x}$ at the i-th ($i < n$) position can be calculated over $\mathcal{M}$ as,

$$\Gamma^k(\mathcal{M}, \mathbf{x}, i) = \sum_{\pi \in \Pi} \rho(\pi)\eta(\pi_r, \hat{x}_{i+k}, \hat{y}),$$

where the source state of $t_i(\mathbf{x})$ is s_i, which maps to abstract state $\hat{s}_i$, the output of $t_{i+k}(\mathbf{x})$ maps to the abstract output $\hat{y}$, and $\Pi(\hat{\mathbf{x}}[i\colon i+k], \hat{s}_i)$ (simply noted as Π) is the set of traces over $\mathcal{M}$ initiating from $\hat{s}_i$ and triggered by the abstract input sub-sequence $\hat{\mathbf{x}}[i\colon i+k]$.

The above definition is consistent with our 0-step aggregated pointwise robustness defined under the abstract domain in Eq. (2). In that case, the sub-sequence $\hat{\mathbf{x}}[i\colon i+k]$ becomes ϵ, thus $\Pi(\epsilon, \hat{s}_i)$ contains only *one* trace which consists of a single self-transition of $\hat{s}_i$ via ϵ, and we have $\Gamma^k(\mathcal{M}, \mathbf{x}, i) = \eta(\hat{s}_i, \hat{x}_i, \hat{y})$.

4 MODEL REFINEMENT STRATEGY

With RNN abstracted as a labeled MDP, we hope to calculate the robustness over the abstraction for better efficiency. To derive an abstraction which allows for accurate estimation of the robustness as obtained from mutation testing (see Definition 3.1), we design a refinement process to reduce the estimation errors iteratively. The aim is to reduce the estimation errors from the k-step aggregated pointwise robustness and gradually approaches the ground-truth robustness reflected by mutation testing. In the following, we assume appropriate input and output abstraction functions are predefined, and present the refinement algorithm focusing on the refinement of the state abstraction function in Section 4.2.

4.1 Input and Output Abstraction

Input abstraction aims to gather similar input elements, ideally the ones which are able to reveal a similar level of robustness at the same states. As a heuristic, we group inputs with identical or similar mutation probability distributions together to form an abstract input. This way, the input abstraction is aligned with the pointwise mutation function (Definition 2.4) and inputs with similar semantics and mutants are grouped and treated equivalently in the MDP abstract model. The mutation functions are designed according to the specific properties of input domains, considering factors such as, whether the domain is continuous (*e.g.*, speech audios) or discrete (*e.g.*, natural languages) and which types of distance measures (*e.g.*, l_p norm distance or cosine distance) are more suitable in restricting the mutation magnitude. Taking the natural language processing (NLP) application as an example, it is natural to consider synonyms as mutants of a word, which maintains semantic similarities. Practically, words with cosine distances within a predefined threshold can be identified as synonyms. We assume the mutation probability of a word to all its synonyms uniformly distributed. Hence, different words sharing the same or similar set of synonyms can be mapped into the same abstract input. The benefit of this choice is that the robustness measures computed for one instance within the synonyms can be more easily generalized for other instances.

For the output abstraction function, it should be designed specifically for different types of deep learning tasks. Similarly, the abstraction on the output domain aims to gather similar outputs together, based on how tolerant the users are about the output variations. For classification tasks, identity function can be used for the output abstraction, since any classification result other than the truth label is regarded as a failure. For tasks attempting to predict a value from continuous domain, *e.g.*, the steering angle of autonomous cars, an abstraction function can be designed to map the outputs to a finite discrete domain with techniques such as predicate abstraction [53]. The predictions of such tasks are deemed as correct as long as they are within a certain range of the truth label. In general, the output abstraction functions can be derived for different domains.

Similarity analysis of inputs/outputs can be a fundamental task when conducting robustness analysis, and several metrics have been proposed and widely used, *e.g.*, p-norm and cosine distance. For different DL applications, we could select suitable similarity metrics. The quality of the input/output abstraction could make a significant difference on the accuracy of MARBLE. For example, a coarse input abstraction grouping inputs that are dramatically different is hard to be refined to be accurate for each input element. In this work, we take the widely used metric (*e.g.*, cosine distance) to perform the input/output abstraction on the selected tasks, and leave the more comprehensive study as future work.

4.2 Refinement of State Abstraction

We propose an algorithm for refining the state abstraction, which aims to reduce the gap between the k-step aggregated pointwise robustness (Definition 3.4) and the individual k-step pointwise robustness (Definition 2.5) within an abstract state. From Definition 3.4, we see that the k-step aggregated pointwise robustness can be deduced from the robustness labeling function.

Definition 4.1 gives the *mean squared error* (MSE) [34] of the estimation calculated using an MDP abstraction compared with the pointwise robustness obtained by mutating individual concrete inputs. The accuracy of the MDP abstraction in measuring robustness is determined by how concrete states are clustered, because errors arise when the aggregated robustness deviates from the robustness observed concretely. Ideally, we would like to design the state abstraction function such that states with similar robustness distributions are clustered into an abstract state. Maximum Mean Discrepancy (MMD) [49, 50] is widely used to measure the distance between distributions, and defined as,

$$MMD(d_1, d_2) = \left\| \mathbb{E}_{X \sim d_1}[X] - \mathbb{E}_{Y \sim d_2}[Y] \right\|_{\mathcal{H}},$$

where d_1 and d_2 are two distributions and the MMD takes the distance between the mean of the two distributions in the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$. Here we take the Euclidean distance with Euclidean space being a widely-used member of RKHS. In Definition 4.1, we sum up the squared MMD distance (in Euclidean space) between all aggregated robustness and concrete robustness pairs to represent the errors introduced by the abstraction.

Definition 4.1 (Mean Squared Error). Given an RNN $\mathcal{R}$, a set of samples D, let $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ and $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta)$ be the FST and the labeled MDP established accordingly. The mean squared error is calculated as,

$$\frac{1}{|\delta|} \sum_{\hat{s} \in \hat{\mathcal{S}}} \left(\sum_{(\hat{s}, \hat{x}, \_) \in \hat{\delta}} \left(\sum_{s \in \hat{s} \wedge x \in \hat{x} \wedge (s, x, \_, \_) \in \delta} \left\| \gamma^0(s, x) - \eta(\hat{s}, \hat{x}) \right\|^2 \right) \right),$$

where $s \in \hat{s}$ and $x \in \hat{x}$.

Next, we elaborate on how to refine a state abstraction function for a given MSE threshold θ, and sketch the process in Algorithm 1. Given an RNN model $\mathcal{R}$, a mutation function μ, a threshold θ and a set of samples D, the algorithm produces a labeled MDP, with which the estimated 0-step aggregated pointwise robustness of samples in D achieves an estimation error less than θ. Here, the pointwise robustness computed by mutation testing with the mutation function μ is used as the ground truth. We assume that λ_I and λ_O are predefined by users; and λ_S is initialized such that all concrete states are mapped to a single abstract state.

Firstly, we profile the RNN model $\mathcal{R}$ with samples in D and represent the traces as an FST, $\mathcal{T}_{\mathcal{R}}(D)$ (Line 3). We then conduct mutation sampling by replacing the input of each transition with N mutants (Lines 4 to 9). The mutants are sampled according to the probabilistic distribution defined by μ. Via counting the frequencies of emitting different outputs after applying the replacements, we obtain the 0-step robustness (a probabilistic distribution over the output domain) for each transition.

Lines 10 to 26 show the core refinement steps. In each iteration, we attempt to improve the accuracy of the abstraction function

Algorithm 1: Refinement algorithm for a labeled MDP.

```
   input : R = (X, S, O, f): RNN, μ: mutation function, θ: threshold, D: samples
   output: M: labeled MDP
1  Prepare input/output/state abstraction functions λ_I, λ_O, λ_S;
2  Mutation sampling count N;
3  T_R(D) = (X, S, O, s_0, F, δ) ;      // finite state transducer of R
4  for (s, x, y, s') ∈ δ do             // mutation sampling for robustness estimation
5      γ^0(s, x) ← [0]^|O|;
6      for j in [0, N) do
7          x'_i ← selectMutation(μ, x_i);
8          (_, y') ← f(s, x'_i);
9          γ^0(s, x)[y'] += 1/N
10 do
11     M = (X̂, Ŝ, Ô, ŝ_0, δ̂, η) ← build_mdp(T_R(D), λ_I, λ_O, λ_S);
12     refinable ← False;
13     for ŝ ∈ Ŝ do                     // refine each abstract state
14         for (ŝ, x̂, _) ∈ δ̂ do         // refine the first refinable transition
15             S_pw, B_pw ← ∅, ∅ for (s, x, _) ∈ (ŝ, x̂, _) do
16                 b ← [0]^|Ô|;
17                 for ŷ ∈ Ô do          // pointwise robustness
18                     b[ŷ] = Σ_{y∈ŷ} γ^0(s, x)[y];
19                 S_pw, B_pw ← S_pw ∪ s, B_pw ∪ b;
20             if 1/|B_pw| Σ_{b∈B_pw} ||η(ŝ, x̂) − b||^2 > θ then   // to refine
21                 K ← fitKmeansCluster(B_pw, clusters = 2);
22                 C ← fitSVMClassifier(S_pw, K.labels);
23                 λ_S ← addSubAbstracter(λ_S, ŝ, C);
24                 refinable ← True;
25                 break
26 while refinable;
27 return M;
```

for each abstract state in $\mathcal{M}$. As the first step, a labeled MDP is computed from $\mathcal{T}_{\mathcal{R}}(D)$ with the input, output, and state abstraction functions. For each abstract input element $\hat{x}$, triggering a transition sourcing from $\hat{s}$ (Line 14), we examine whether the estimation error of this pair is below θ (Line 20). We calculate the mutation-based robustness related to $\hat{s}$ and $\hat{x}$ with Lines 15 to 19, and store the results in B_{pw}. We also keep the concrete state vectors in S_{pw}, one-to-one mapped to those robustness in B_{pw}. The 0-step aggregated pointwise robustness estimated for the $(\hat{s}, \hat{x})$ pair is $\eta(\hat{s}, \hat{x})$, which equals to the average of the values in B_{pw}. If the MSE of this subset of observations is greater than θ (Line 20), we make a refinement on $\hat{s}$ such that the overall MSE on D is reduced (Theorem 4.3).

We design a two-step refinement strategy to narrow the distance between the estimated aggregated robustness and every corresponding mutation testing robustness. We aim to cluster the concrete states in S_{pw} according to their mutation-based robustness (Line 21), such that states with similar robustness would be gathered together. In the first step, we leverage k-means to separate the robustness distributions in B_{pw} into two clusters. On Line 21, $\mathcal{K}$ denotes the fitted k-means classifier, and $\mathcal{K}.labels$ is used to represent the list of obtained cluster labels. Clustering over the robustness distributions offers implication on how to divide the state space. In the second step, we treat the cluster index as the label and employ SVC classifier C to approximate the decision boundary over the state space (Line 22). Finally, we append C as a sub-abstraction function of state $\hat{s}$ and update the state abstraction function λ_S (Line 23). After

examining all abstract state and input element pairs, if there exists any pair to be refined, we re-build the MDP $\mathcal{M}$ with the updated λ_S and continue the iteration. Otherwise, if neither of the abstract states requires further refinement, the procedure terminates and returns the refined MDP model.

Example 4.2. Following Fig. 3(b), we illustrate how the refinement algorithm works to reduce the estimation error. We assume a threshold of $\frac{1}{27}$ on the estimation error. When the loop iterates at $(\hat{s}_1, \hat{x}_4, \_)$ at Line 14, the list of concrete states is $S_{pw} = [s_3, s_6, s_{10}]$, and the corresponding list of pointwise robustness observations is $B_{pw} = [(\frac{1}{3}, \frac{2}{3}), (\frac{1}{3}, \frac{2}{3}), (\frac{2}{3}, \frac{1}{3})]$. As we know from the labeling function, the 0-step aggregated robustness at $\hat{s}_1$ over $\hat{x}_4$ is $\eta(\hat{s}_1, \hat{x}_4) = (\frac{4}{9}, \frac{5}{9})$. The local estimation error yields as,

$$\frac{1}{3}\left(\left\|(\tfrac{1}{3}, \tfrac{2}{3}) - (\tfrac{4}{9}, \tfrac{5}{9})\right\|^2 + \left\|(\tfrac{1}{3}, \tfrac{2}{3}) - (\tfrac{4}{9}, \tfrac{5}{9})\right\|^2 + \left\|(\tfrac{2}{3}, \tfrac{1}{3}) - (\tfrac{4}{9}, \tfrac{5}{9})\right\|^2\right) = \frac{4}{81} > \frac{1}{27},$$

indicating a further refinement is required.

First, we apply k-means over B_{pw}, and set the expected cluster number to two. Since k-means aims to put closer points into the same cluster, it is likely to return two clusters $C_0 = \left[\left(\frac{1}{3}, \frac{2}{3}\right), \left(\frac{1}{3}, \frac{2}{3}\right)\right]$ and $C_1 = \left[\left(\frac{2}{3}, \frac{1}{3}\right)\right]$. Hence the labels for the list of states are $\mathcal{K}.labels = [C_0, C_0, C_1]$. Then, we can fit the SVC which splits the abstract state $\hat{s}_1$ into two. As demonstrated in Fig. 3(c), we assume the red dashed curve to be the boundary determined by the SVC. Thus, the original abstract state $\hat{s}_1$ is refined into two new abstract states $\hat{s}_5$ and $\hat{s}_6$. Now, the state abstraction is refined, and the MDP model is to be reconstructed.

Theorem 4.3 (Error Reduction). *Given an RNN $\mathcal{R}$, a set of samples D, let $\mathcal{M}$ and $\mathcal{M}'$ be labeled MDPs obtained from $\mathcal{R}$ and D, before and after the execution of the refinement step (Lines 21 to 25 in Algorithm 1). We have $MSE_{\mathcal{M}'} \leq MSE_{\mathcal{M}}$, where $MSE_{\mathcal{M}}$ and $MSE_{\mathcal{M}'}$ are the mean squared errors of $\mathcal{M}$ and $\mathcal{M}'$, respectively.*

The proof of Theorem 4.3 is available on our website [19]. With Theorem 4.3, the MSE of $\mathcal{M}$ is reduced after each refinement step. Since there is only a finite number of concrete states in $\mathcal{S}$, and the number of abstract states $\hat{\mathcal{S}}$ is guaranteed to increase after each iteration, $MSE_{\mathcal{M}}$ is eventually approaching zero. Therefore the threshold will be reached after finite number of iterations.

5 EVALUATION

We implemented Marble in Python based on the PyTorch (1.2.0) framework and conducted evaluation on six real-world RNN subject models to evaluate the refinement strategies and the robustness quantification method. Specifically, our experiments are designed to answer the following research questions:

- **RQ1:** How effective is the refinement algorithm in generating MDP abstractions? How are the abstractions in terms of succinctness and generalization?
- **RQ2:** How is the scalability and efficiency of the robustness quantification by Marble?
- **RQ3:** Does Marble provide a better quantification of RNN robustness than state-of-the-art approaches?

Subject Datasets and RNN Models We selected three popular NLP datasets and trained the corresponding RNN models. In particular, we leveraged the pre-trained word embedding vectors GloVe [44] to accelerate the training process and achieve competitive accuracy.
(1) The CogComp QC Dataset (abbrev. QC) [37] includes news titles labeled with different types of topics. There are 20K samples for training and 8K samples for testing, and each sample contains 9.2 words on average. We followed the same configurations as in [33] to train an LSTM model with a test accuracy of 83.3% and a GRU model with a test accuracy of 83.0%, respectively.
(2) The Jigsaw Toxic Comment Dataset (abbrev. Toxic) [1] used in Kaggle challenge includes a set of comments from Wikipedia's talk page edits and is labeled as toxic or not, with an average sample length of 54.8. The original dataset is designed to be severely imbalanced for the challenge usage. Here in order to obtain an accurate model, we use 25k non-toxic samples and 25k toxic samples for the model training (80%) and testing (20%). We trained an LSTM model (with 92.7% test accuracy) and a GRU model (with 93.1% test accuracy), with 128 hidden nodes respectively.
(3) The Sentiment Analysis Dataset (abbrev. IMDb) [39] contains IMDb movie reviews labeled with binary (positive or negative) sentimental classifications. There are 25K training data and 25K test data, where each sample contains 255.8 words on average. We trained an LSTM model and a GRU model, each with 300 hidden nodes, which achieve 90.7% and 90.9% test accuracy respectively.

States Preprocess and Input Abstraction. To handle high-dimensional state vectors, we applied Principal Component Analysis [56] to reduce the data dimension to ω major components as in [20]. To build the mutation function for NLP applications, we use *cosine similarity* to measure the semantic distances between words. Based on the GloVe embedding vector, we performed the input abstraction by making synonym groups, among which the cosine similarities are above a threshold. In our evaluation, we set the a higher threshold (0.75) to guarantee a conservative metamorphic relation and obtained 3,572 synonym groups. During the mutation, we allowed a uniform probability distribution on words within the same synonym group.

5.1 RQ1: Effectiveness of the Refinement

We first examined the performance of our refinement strategy under different configuration instances (ω, θ), where ω is the retained dimension of the state vectors after PCA and θ is the MSE threshold. As a baseline for comparison, we also implemented a random-split based strategy for abstract state refinement. The random-split strategy differs from Marble at only two steps (cf. Line 21 and Line 22 in Algorithm 1), which refines an abstract state by randomly separating the state instances into two groups, and fit the SVC classifier.

Our evaluation was conducted on the six RNN models (both LSTM and GRU) of the three datasets. For each RNN model, we experimented on six (ω, θ) configurations. Table 1 summarizes the results of refinement obtained from the LSTM models of the three datasets, under three configurations. The full results of all models are on our website.[1] Column "Config." shows the evaluated configuration instances (*i.e.*, (ω, θ)), which are used in both of Marble and the random strategy. The other columns include the number

[1] https://sites.google.com/view/marble-rnn

Table 1: Measures of MDP Models Refinement.

	Config.	Strategy	#Iter.	Time (s)	#State	#Tran.	MSE_t	Miss (%)
QC	(16, 0.3)	Marble	22	60.4	1158	136100	0.04	45.31
		Random	43	49.8	3149	168552	0.06	56.68
	(16, 0.6)	Marble	20	89.4	864	129049	0.05	41.07
		Random	31	54.1	2414	164461	0.06	55.59
	(16, 1.0)	Marble	16	112.6	447	110649	0.07	33.34
		Random	25	56.1	1228	157208	0.09	50.47
Toxic	(16, 0.3)	Marble	23	947.1	3577	859443	0.01	13.73
		Random	162	649.6	12997	1147488	0.01	22.68
	(16, 0.6)	Marble	19	969.8	2083	729806	0.01	11.23
		Random	105	763.9	10021	1118550	0.01	20.36
	(16, 1.0)	Marble	23	834.5	973	604829	0.01	8.34
		Random	88	645.4	4782	1028458	0.02	17.44
IMDB	(32, 0.3)	Marble	31	2849.7	13494	4075323	0.02	25.20
		Random	127	805.1	67162	6472855	0.06	49.90
	(32, 0.6)	Marble	30	2028.7	8344	3900624	0.03	21.89
		Random	42	812	53392	6426118	0.06	46.88
	(32, 1.0)	Marble	26	2385.7	3541	3437900	0.03	16.80
		Random	38	812	23510	6032349	0.08	38.88

of iterations taken by the refinement ("#Iter"), the refinement time ("Time (s)"), the number of states ("#State."), the number of transitions ("#Tran.") in the refined MDP. Further, we examined the generalization ability of the labeled MDP when applied to the robustness estimation on the never seen test data, and reported the percentage of missed transitions in column "Miss (%)" as well as the *MSE* measure in column "MSE_t". We refer to a transition as missed if it is not included in the MDP model. *MSE* is calculated in a similar way as when building the MDP model, where we profiled the test data, obtained the mutation-based robustness, and compared them with the estimation by the labeled MDP.

The results confirm the advantages of the refinement strategy of Marble compared with a random-split approach in that: **1)** Marble terminates with less iterations across configurations and datasets. For the three datasets, the average number of iterations taken by Marble is 19.3, 21.7 and 29.0, while that by random strategy is 53.5, 118.3 and 69.0. **2)** Marble always derives a smaller MDP model with less states and transitions through the refinement. For example, the average number of states for the 3 datasets is 823.0, 2211.0 and 8459.7, respectively, while that by random strategy is 2263.7, 9266.7 and 48021.3, which are 1.8, 3.2 and 4.7 times larger than Marble. **3)** Marble always derives a better MDP model, which demonstrates not only a less transition miss rate when dealing with new samples from the test dataset, but also a smaller estimation error. For instance, the average miss rate on the 3 datasets is 39.9%, 11.1% and 21.3%, while the random strategy causes larger miss rate with respectively 54.2%, 20.2% and 45.2%. However, as a trade-off, Marble takes longer time to complete the refinement. For example, the average time used is 87.5s, 917.1s and 2421.4s, which are larger than that by the random strategy, with respectively 53.3s, 686.3s and 809.7s. This is due to the application of the k-means, which better refines the abstract state with robustness estimation.

Finally, we summarize implications on applying Marble to dataset of different scales and with various configurations. Basically, under the same configuration, it takes more time and iterations to complete the refinement of larger dataset and more complicated RNN models, and the resulted MDP is also larger. Under the same configuration of ω, the larger threshold θ is, the less effort it takes to complete the refinement, resulting in a less complicated MDP, but with slightly higher estimation error. A coarse MDP often endows better generalization capability when processing new samples, leading to less missed transitions.

> **Answer to RQ1:** Marble is superior than the random-split refinement strategy, and can deliver a more accurate abstract MDP model with smaller size and better generalization ability. Marble takes less number of iterations but costs slightly more time to accomplish the abstraction.

5.2 RQ2: Performance Efficiency

In this experiment, we evaluate the time taken to quantify the robustness over every element of a sample, the efficiency of which can be highly desirable for real-time monitoring applications. We present the results of three LSTM models, and full results can be found on website [5].

For each of the refined MDP, we apply it to estimate the 0-step robustness of samples in the test data, and report the average time used per sample in Table 2. Note that we take the total time used to get the 0-step robustness for every element in a sample, which indicates that the longer the sample, the more time would be used. In Table 2, the first column shows the dataset and the second column "Config." presents the (ω, θ) configurations used to obtained the MDP, and the time used by the robustness estimation is given in column "Time (s)". Overall, the robustness of an input can be calculated efficiently by Marble, with the time magnitude of thousands or dozens of millisecond. On average, the three datasets take 0.03s, 0.24s and 1.07s to estimate the robustness of an individual sample. Larger dataset tends to take more time to finish the estimation.

We compare the scalability and efficiency of Marble with the state-of-the-art robustness quantification approach for RNNs, *i.e.*, POPQORN. However, we cannot comapre Marble with POPQORN in all models for two reasons: 1) scalability is a major limitation for POPQORN, which fails to complete on larger RNNs trained from Toxic and IMDb datasets and 2) POPQORN is model-dependent and the current released version does not support GRU. Thus, we only run POPQORN on the LSTM model of QC dataset.

We ran POPQORN on all sentences from the test dataset to calculate their robustness scores and set a timeout, *i.e.*, 12 hours. Finally, only the robustness score of 42 sentences are calculated successfully. On average, POPQORN takes 52.8 minutes to complete the estimation of a single input.

> **Answer to RQ2:** Compared with POPQORN, Marble offers better scalability for handling large and complicated models for accepting longer inputs, and is also efficient in robustness calculation, thus can be applied for real-time robustness monitoring of larger RNN applications.

5.3 RQ3: Accuracy of the Robustness Measures

Due to the absence of the ground truth on the robustness measurement, it is not possible to gauge the absolute estimation accuracy.

Table 2: Attack Success Rate and Estimation Efficiency.

Dataset	Config.	$ASR_l(\%)$			$ASR_m(\%)$			Time (s)
		MARBLE	Vs Random	Vs POPQORN	MARBLE	Vs Random	Vs POPQORN	
QC	(16, 0.3)	20.24	72.41	35.78	5.76	−50.91	-42.11	0.02
	(16, 0.6)	20.00	70.39	34.19	4.40	−62.47	-55.74	0.02
	(16, 1.0)	22.71	93.51	52.40	4.24	−63.89	-57.42	0.02
	(32, 0.3)	19.83	68.97	33.07	2.67	−77.28	-73.21	0.03
	(32, 0.6)	20.10	71.20	34.82	2.57	−78.09	-74.16	0.05
	(32, 1.0)	17.74	51.12	19.01	5.40	−53.96	-45.69	0.06
	Avg.	20.10	71.26	34.88	4.17	−64.44	-58.05	0.03
Toxic	(16, 0.3)	7.27	143.52	-	2.00	−33.11	-	0.22
	(16, 0.6)	8.76	193.59	-	2.29	−23.23	-	0.24
	(16, 1.0)	10.24	243.12	-	2.29	−23.36	-	0.21
	(32, 0.3)	7.80	161.28	-	2.10	−29.77	-	0.27
	(32, 0.6)	10.46	250.47	-	2.24	−25.10	-	0.25
	(32, 1.0)	11.74	293.46	-	2.20	−26.30	-	0.23
	Avg.	9.38	214.24	-	2.18	−26.81	-	0.24
IMDB	(32, 0.3)	4.76	176.51	-	1.11	−35.35	-	1.09
	(32, 0.6)	3.60	109.53	-	1.17	−32.09	-	1.04
	(32, 1.0)	4.01	133.26	-	1.06	−38.37	-	0.84
	(64, 0.3)	4.04	134.88	-	1.06	−38.37	-	1.25
	(64, 0.6)	3.59	108.60	-	1.01	−41.40	-	1.16
	(64, 1.0)	2.79	62.33	-	1.16	−32.79	-	1.05
	Avg.	3.80	120.85	-	1.09	−36.40	-	1.07

* The ASR differences between MARBLE and other approaches are examined with Mann-Whitney U test, and the displayed results are statistically significant with $p < 0.05$.

Instead, we inspected whether the locations witnessing worse robustness is more vulnerable to attack, *i.e.*, to examine relatively order the robustness estimations. Worse robustness indicates that perturbations over that location can easily make adversarial attacks. We evaluated the effectiveness of the obtained robustness measures by examining their helpfulness in launching adversarial attacks. According to the robustness reverted by the quantification tools, attacking the least robust locations leads to the most efficient attack. To measure the attack efficiency, we assigned a fixed number of attack chances to each sample, and calculated the attack success rate (ASR) over a set of samples, *i.e.*, the portion of samples that are successfully manipulated into adversarial. Our evaluation was conducted on all the refined MDPs of the LSTM and GRU models, and with the test dataset. We select two baselines: 1) random strategy that randomly selects a location to mutate and 2) robustness-guided attack based on the measurements by POPQORN.

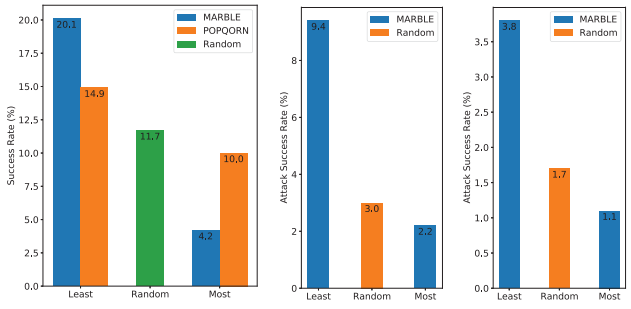
For the QC dataset, since POPQORN is only successfully executed on 42 samples, we used the same set of samples to evaluate the MDPs refined from the QC LSTM model to make a fair comparison. Specifically, for each sentence, we performed two sets of experiments to attack the least robust word and the most robust word, respectively. We replaced the selected word to one of its synonyms in a random manner and the attack was claimed to be successful whenever the prediction result became different.

We repeated the experiments 100 times and report the average ASR. An overall comparison among these three approaches is demonstrated as a bar plot in Fig. 4. For each dataset, it displays the average performance of MARBLE calculated over all abstractions, the performance of POPQORN as well as the random attack strategy.

We also present the detailed results of MARBLE under various configurations in Table 2. The three columns under "$ASR_l(\%)$" reports the ASR when attacking the least robust locations and the increase rate of our approach when respectively compared with the random strategy and POPQORN, while the results of attacking the most robust locations are under "$ASR_m(\%)$". Here the ASR values are averaged over the 100-time executions. The performance difference is also statistically significant (*i.e.*, confirmed by Mann-Whitney U test [40] at $p < 0.05$ confidence level).

From Fig. 4, we can see that the random attack strategy makes a success rate of 11.7%, 3.0% and 1.7%, respectively on the 3 dataset. Compared with random strategy, MARBLE achieves higher ASR (*i.e.*, on average 20.1%, 9.4% and 3.8%) when attacking at the least robust locations and lower success rate (*i.e.*, on average 4.2%, 2,2%, 1.1%) when attacking the most robust locations. In the best cases, as shown in Table 2, MARBLE outperforms the random strategy with an ASR increase of 93.51%, 293.46% and 176.51%, when attacking the least robust locations. For attacks on the most robust locations, the ASR by MARBLE can be 77.28%, 33.11% and 41.40% less than that of the random strategy. On the QC dataset, POPQORN achieves an ASR of 14.9% when attacking the least robust locations, which is only 27.4% higher than the random strategy and worse than all the results by MARBLE. When attacking the most robust locations, POPQORN still raises a high ASR of 10%, while MARBLE only makes 4.2%.

Answer to RQ3: In our evaluation, MARBLE calculates robustness more accurately than POPQORN. MARBLE can also achieve up to 2 times higher attack success rate than the random strategy.



(a) QC dataset. (b) Tosic dataset. (c) IMDb dataset.

Figure 4: Attack success rates based on the robustness scores calculated by different methods.

5.4 Threats to Validity

First, six subject RNN models and a limit set of configurations were examined for the state abstraction refinement. We have tried our best to cover a wide range of MSE thresholds, which is an important and direct factor influencing the accuracy of the robustness estimation. Even with this, conclusions drawn on the limited group of subjects may not generalize to other models and configurations. Second, for the attacking experiments aiming to evaluate the model accuracy, the set of samples to be attacked is of a relatively small size. For QC dataset, to accommodate the scalability of POPQORN, only 42 samples were investigated. Hence, the comparison results with POPQORN may lack statistical significance. Finally, randomness is hard to be avoided for both the refinement process and the attacking experiments. The initial cluster centers of k-means were randomly selected, thus the state abstraction achieved may vary between executions. Moreover, mutations were randomly selected (according to the mutation probability distribution) during the attack procedure. Given a fixed number of chances, whether we can construct an adversarial sample is non-deterministic. In order to offset the randomness (to a certain extend), we repeated all the experiments for five times and reported the average values.

6 RELATED WORK

Adversarial Attacks on RNNs NLP and automatic speech recognition are the two typical domains where existing RNN robustness attack applies. To attack NLP models, paper [42] is an early work to launch adversarial attacks on text classification task with Fast Gradient Sign Method. Paper [36] provided a technique to locate important and sensitive words with reinforcement learning. Following the similar line, paper [31] proposed to score each word with an importance metric and leverage word manipulations, including swap, substitution, deletion and insertion, to generate adversarial examples for reading comprehension systems. For text classification models, there have also been works [13, 21] on generating adversarial examples to fool general-purpose sequence-to-sequence models.

For attacks of speech recognition models, research work [26] presented an approach to generate untargeted audio attacks, while paper [15] proposed a technique to generate phonetically similar phrases and made it possible to generate targeted attacks. Paper [11] further advanced the adversarial audio generation to produce imperceptible attacks for any given targets and evaluate their approach on popular model DeepSpeech [2]. The after-mentioned approaches all reply on gradient-based algorithms, which require the white-box information on the full parameters of the subject models. A GAN-based black-box attack generation is designed in [57] with the demonstrations on textual entailment and machine translation for untargeted attacks. Analyzing the difficulties of leveraging existing attacks to RNN through either direct application or transfer attacks enables to estimate robustness of an RNN against an input.

Robustness Analysis of DNN Instead of answering whether there exists an attack to compromise the DNN, robustness verification aims to identify the minimum adversarial distortion bound such that no attack exists with distortion under this lower bound. Existing works along this direction make some progress to verify the robustness of FNNs. The minimum adversarial distortion calculation of ReLU networks is shown to be NP-hard [32]. Therefore, further works attempt to compute a non-trivial certified lower bound [9, 48, 54] or to find an estimation of the minimum adversarial distortion [7, 55]. Bastani *et al.* [7] proposed to encode the DNN as a linear programming problem, and defined a robustness metric based on the adversarial examples discovered. This metric depicts an upper bound of the minimum distortion and also dependents on specific attack algorithms. In contrast, the CLEVER score [55] provides an attack-agnostic estimation of the lower bound. Paper [24] developed the first sound analyzer for DNN with abstract interpretation [16, 17], to automatically prove robustness properties.

To our best knowledge, the only research on the robustness verification of RNN is POPQORN [33], which proposes a robustness quantification framework to develop a guaranteed lower bound of the minimum distortion in RNN attacks. However, it suffers from scalability issues, which cannot handle hundreds of hidden layers as demonstrated in our empirical evaluation.

7 CONCLUSION

This paper proposed MARBLE, a model-based technique for quantitative robustness analysis of RNN-based DL systems. Our evaluation MARBLE enables more accurate and efficient quantification of the robustness of RNNs. In the future, we plan to conduct studies on the effect parameters k (in the k-step trace-based robustness) on the robustness quantification and apply MARBLE on diverse practical applications, such as image classification and automatic speech recognition.

ACKNOWLEDGMENTS

This work was supported by Singapore Ministry of Education Academic Research Fund Tier 1 (Award No. 2018-T1-002-069), the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), the Singapore National Research Foundation under NCR Award Number NSOE003-0001, NRF Investigatorship NRFI06-2020-0022 and NTU GAP funding (NGF-2019-06-024). It was also supported by JSPS KAKENHI Grant No. 20H04168, 19K24348, 19H04086, and JST-Mirai Program Grant No. JPMJMI18BB, Japan. We also gratefully acknowledge the support of NVIDIA AI Tech Center (NVAITC) to our research.

REFERENCES

[1] 2018. Jigsaw Toxic Comment Classification Challenge. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

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

[3] 2020. Amazon Alexa. https://developer.amazon.com/alexa

[4] 2020. Categorical Distribution. https://en.wikipedia.org/wiki/Categorical_distribution

[5] 2020. MARBLE home page. https://sites.google.com/view/marble-rnn/home

[6] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. 2019. Certifying Geometric Robustness of Neural Networks. In *Advances in Neural Information Processing Systems 32*.

[7] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., USA, 2621–2629. http://dl.acm.org/citation.cfm?id=3157382.3157391

[8] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 387–402.

[9] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. 2019. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3240–3247.

[10] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), IEEE Symposium on*. 39–57.

[11] 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

[12] Guangke Chen, Sen Chen, Lingling Fan, Xiaoning Du, Zhe Zhao, Fu Song, and Yang Liu. 2019. Who is Real Bob? Adversarial Attacks on Speaker Recognition Systems. *arXiv preprint arXiv:1911.01840* (2019).

[13] Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. 2018. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *arXiv preprint arXiv:1803.01128* (2018).

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

[15] Moustapha Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. 2017. Houdini: Fooling Deep Structured Prediction Models. (jul 2017). arXiv:1707.05373 http://arxiv.org/abs/1707.05373

[16] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 238–252.

[17] Patrick Cousot and Radhia Cousot. 1992. Abstract interpretation frameworks. *Journal of logic and computation* 2, 4 (1992), 511–547.

[18] Xiaoning Du. 2019. Towards secure and robust stateful deep learning systems with model-based analysis. (2019).

[19] Xiaoning Du, Yi Li, Xiaofei Xie, Lei Ma, Yang Liu, and Jianjun Zhao. 2020. Supplementary Materials for 'Marble: Model-based Robustness Analysis of Stateful DeepLearning Systems'. https://doi.org/10.21979/N9/TTTSFK

[20] 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 *FSE*. 477–487.

[21] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751* (2017).

[22] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust Physical-World Attacks on Deep Learning Visual Classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. 1625–1634.

[23] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 50–56.

[24] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

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

[26] Y. Gong and C. Poellabauer. 2017. Crafting Adversarial Examples For Speech Paralinguistics Applications. *ArXiv e-prints* (Nov. 2017). arXiv:1711.03280

[27] 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

[28] Theodore P. Hill and Jack J. Miller. 2011. How to Combine Independent Data Sets for the Same Quantity. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 21, 3 (2011).

[29] 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.

[30] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A Mutation Testing Framework for Deep Learning Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1158–1161.

[31] Robin Jia and Percy Liang. 2017. Adversarial Examples for Evaluating Reading Comprehension Systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2021–2031.

[32] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[33] Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019. POPQORN: Quantifying Robustness of Recurrent Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, California, USA, 3468–3477. http://proceedings.mlr.press/v97/ko19a.html

[34] Erich L Lehmann and George Casella. 2006. *Theory of point estimation*. Springer Science & Business Media.

[35] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271* (2018).

[36] Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220* (2016).

[37] Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1–7.

[38] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 120–131.

[39] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics, 142–150.

[40] 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.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[42] Nicolas Papernot, Patrick Drew McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting adversarial input sequences for recurrent neural networks. In *35th IEEE Military Communications Conference, MILCOM 2016*. Institute of Electrical and Electronics Engineers Inc., 49–54.

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

[44] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[45] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[46] 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.

[47] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

[48] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*. 10802–10813.

[49] Bharath K. Sriperumbudur, Kenji Fukumizu, and Gert R. G. Lanckriet. 2011. Universality, Characteristic Kernels and RKHS Embedding of Measures. *J. Mach. Learn. Res.* 12, null (July 2011), 2389–2410.

[50] Bharath K. Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert R. G. Lanckriet. 2009. Hilbert Space Embeddings and Metrics on Probability Measures. *J. Mach. Learn. Res.* 11 (2009), 1517–1561.

[51] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).
[52] The BBC. 2016. AI image recognition fooled by single pixel change. https://www.bbc.com/news/technology-41845878
[53] Bjorn Wachter, Lijun Zhang, and Holger Hermanns. 2007. Probabilistic model checking modulo theories. In *fourth international conference on the quantitative evaluation of systems (QEST 2007)*. IEEE, 129–140.
[54] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. 2018. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699* (2018).
[55] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. 2018. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578* (2018).
[56] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.
[57] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2017. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342* (2017).