RNNRepair: Automatic RNN Repair via Model-based Analysis

Xiaofei Xie [1 2] **Wenbo Guo** [3] **Lei Ma** [4 5 2] **Wei Le** [6] **Jian Wang** [1] **Lingjun Zhou** [7] **Xinyu Xing** [3] **Yang Liu** [1]

Abstract

Deep neural networks are vulnerable to adversarial attacks. Due to their black-box nature, it is rather challenging to interpret and properly repair these incorrect behaviors. This paper focuses on interpreting and repairing the incorrect behaviors of Recurrent Neural Networks (RNNs). We propose a lightweight model-based approach (*RNNRepair*) to help understand and repair incorrect behaviors of an RNN. Specifically, we build an influence model to characterize the stateful and statistical behaviors of an RNN over all the training data and to perform the influence analysis for the errors. Compared with the existing techniques on influence function, our method can efficiently estimate the influence of existing or newly added training samples for a given prediction at both sample level and segmentation level. Our empirical evaluation shows that the proposed influence model is able to extract accurate and understandable features. Based on the influence model, our proposed technique could effectively infer the influential instances from not only an entire testing sequence but also a segment within that sequence. Moreover, with the sample-level and segment-level influence relations, *RNNRepair* could further remediate two types of incorrect predictions at the sample level and segment level.

1. Introduction

In spite of many state-of-the-art applications and high test accuracy, Deep Neural Networks (DNNs) still make mistakes and output wrong predictions. To fix an incorrect prediction, it is important to understand the root cause of the wrong prediction (Koh & Liang, 2017). Once the root cause is identified, users may fix the errors by removing harmful training data or adding specific data to improve model accuracy (Hara et al., 2019). However, due to the black-box nature of the DNN, it is challenging to identify the "most responsible" training samples and explain the wrong predictions. As a result, wrong predictions are difficult to be corrected. Recently, influence functions have been widely studied for interpreting the predictions of DNNs by estimating the effect of removing training samples (Koh & Liang, 2017; Khanna et al., 2019; Koh et al., 2019; Hara et al., 2019). However, using it as a method to remediate misclassification or wrong prediction is still challenging.

First, existing influence-function based methods are mostly designed for feed-forward neural networks (FNNs). Given Recurrent Neural Networks (RNNs), they usually suffer from the vanishing gradient and long-distance dependency. As a result, existing techniques could not be easily applied for RNNs . Second, different from FNNs, RNNs often come with *stateful* structures for processing sequential inputs (*e.g.*, audio, natural language). For a sequential test input, we need to study the effect of its segments more precisely. For example, in automatic speech recognition, we want to identify which training samples are most responsible for the poor recognition of a specific pronunciation (*i.e.*, segment). Existing methods mainly performed influence analysis for the *whole test input* but not at the *segment level*. Third, to use influence analysis based interpretation for repairing a wrong prediction, one needs to select helpful samples from a large number of collected or generated new samples. However, existing influence analysis based methods inevitably introduce intensive computation, making the selection of useful samples inefficient. As a result, it greatly limits their potentiality to be used as a mechanism to repair the errors of a model.

In this work, we propose a light-weight model-based influence analysis for RNNs, named *RNNRepair*[1]. To capture the stateful behaviors of training data, we first construct an influence model from concrete prediction traces of all training data. This model extracts accurate features of inputs from RNNs via state clustering. We then calculate the influential training samples for the segment of an input under

[1]Nanyang Technological University, Singapore [2]Kyushu University, Japan [3]College of Information Sciences and Technology, The Pennsylvania State University, State College, PA, USA [4]University of Alberta, Canada [5]Alberta Machine Intelligence Institute, Canada [6]Iowa State University, USA [7]Tianjin University, China. Correspondence to: Xiaofei Xie <xiaofei.xfxie@gmail.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]https://bitbucket.org/xiaofeixie/rnnrepair

a state (*i.e.*, the context of the input). As part of this work, we also demonstrate the utility of the proposed influence analysis in multiple applications, such as ❶ understanding the behaviors of the RNN, ❷ fixing influential mislabeled data, ❸ pinpointing Trojan backdoor in an RNN model, and ❹ repairing incorrect predictions.

2. Related Work

Model Extraction. Existing research has developed various approaches to extract DFA (Deterministic Finite Automaton) from the known RNN architectures. Specifically, early-stage explorations focused on extracting DFA from the second-order RNNs (Omlin & Giles, 1996; Giles et al., 1990; 1992). More recent works extend the preliminary techniques to GRUs and LSTMs, which have higher practicality than the second-order RNNs (Weiss et al., 2018; Cho et al., 2014; Chung et al., 2014; Weiss et al., 2019; Okudono et al., 2019; Ayache et al., 2018; Zhang et al., 2021). In terms of the state vector partition strategy, existing techniques mainly follow two different methods – (1) equipartition-based approach (Omlin & Giles, 1996; Weiss et al., 2018), which divides each dimension of the latent representations into k equal intervals, and (2) unsupervised learning-based approach (Zeng et al., 1993; Cechin et al., 2003), which applies the existing clustering method (*e.g.*, K-means, GMM) to cluster the state vectors into different groups. DeepStellar (Du et al., 2019) extracts a discrete-time Markov chain (DTMC) from an RNN. Then the DTMC is used for the testing and adversarial example detection. As introduced later in Section 3, we follow the second strategy and apply GMM for state partition. Despite using the same partition strategy, our method is fundamentally different from the existing DFA extraction techniques in that none of the existing methods could derive the influence of training samples upon a given testing sample (*i.e.*, influence relations). In addition, to precisely simulate the behaviors of a target RNN, most of the existing methods need to exhaustively search for the state transitions in a target RNN, which limits their scalability in some applications. However, our method only requires a coarse-grained approximation for deriving the influence relationships, which is much lighter-weight than the existing model extraction methods.

Influence Analysis. Koh *et al*. (Koh & Liang, 2017) studied the influence of training samples upon a given testing sample for DNNs. Specifically, they utilized the *influence function* to identify the most representative training samples for a given testing sample. Following (Koh & Liang, 2017), recent efforts have been made to either enable the influence analysis for non-optimal models trained by using non-convex losses (Hara et al., 2019) or analyze the influence of a group of training samples upon a given prediction (Koh et al., 2019). Despite deriving meaningful influence relations for feed-forward networks (*i.e.*, MLP and CNN), the existing methods might not be effective for RNNs due to the infamous gradient vanishing/explosion problem. In this work, our proposed method does not depend on the gradient calculation and could capture the stateful behaviors of the RNN accurately with the state clustering. In addition, our method is much lighter-weight as shown in Section 4. Furthermore, our method could provide a finer-grained influence relation than the existing methods or, in other words, we can, at the segment level, pinpoint the most influential training samples to the segments within a testing sample.

DNN Repair. Wang *et al*. offset errors made by logistic regressions by integrating an additional layer into the model to pre-process the error inputs (Wang et al., 2019). Different from this work, our method does not modify the model architecture and targets more complex networks – RNN. Yu *et al*. proposed a style-guided repair for the unknown failure pattern in DNNs with a style transfer method (Yu et al., 2020). Some works (Sotoudeh & Thakur, 2019; Zhang & Chan, 2019) propose to repair the model by changing the network weights. Differently, our method focuses on repairing the specific failed samples for the RNN with a model-based influence analysis. It should be noted that our method is different from the techniques about adversarial defenses (Boopathy et al., 2019; Weng et al., 2018; Singh et al., 2018) and noisy learning/data cleaning (Zhang et al., 2018) in that these techniques aim for improving the robustness against adversarial attacks or data poisoning attacks. Whereas, our remediation mechanism locally offsets testing errors of an RNN.

3. Approach

Overview. At a high level, we first adopt the clustering to capture the stateful behaviors of all training data. Based on the state transitions, we then identify the influential training samples of a given testing input or a segment of the test input. Specifically, we first extract the abstract states by grouping state vectors (*i.e.*, hidden representations of the RNN) of all training data (Section 3.1). Then, based on the state abstraction, we can extract the trace for a given input. We construct the influence function based on the transitions in the traces and further perform the segment-level and sample-level influence analysis. (Section 3.2). Last but not least, based on the influence analysis, we develop an remediation mechanism to analyze and offset the test errors (Section 3.3).

3.1. Semantic-guided State Abstraction

Definition 1 (RNN) *An RNN is defined as a 5-tuple* $R = (\mathcal{G}_R, d, m, \boldsymbol{h_0}, \mathcal{Y}_R)$*: $\mathcal{G}_R$ is a recursive function* $\boldsymbol{h_t} = \mathcal{G}_R(\boldsymbol{x}_t, \boldsymbol{h_{t-1}})$*, where* $\boldsymbol{h}_t \in \mathbb{R}^d$ *is a d-dimensional state vector,* $\boldsymbol{x}_t \in \mathbb{R}^m$ *is the m-dimensional input vector at time t; d and m are the dimensions of the state vector and the*

input vector, respectively. $\boldsymbol{h}_0 \in \mathbb{R}^d$ *is the initial state; The output function* $\mathcal{Y}_R : \mathbb{R}^d \to \mathbb{R}$ *maps an internal state-vector to the output value.*

Given a sequential input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, an RNN generates a sequence of state vectors $(\boldsymbol{h_0}, \boldsymbol{h_1}, \ldots, \boldsymbol{h_n})$ with the application of $\mathcal{G}_R$. To simplify the notation, we use $\mathcal{G}_R^{\boldsymbol{x}}$ to denote the state vector sequence of the input $\boldsymbol{x}$. $\mathcal{Y}_R$ calculates different types of outputs based on different applications. In this paper, we mainly focus on the classification problem, where the output function maps each state vector to a specific class (*i.e.*, $\mathcal{Y}_R^n : \mathbb{R}^d \to \{0, \ldots, n-1\}$, n is the total number of classes). Specifically, for the sequence classification problem (*e.g.*, semantic analysis), the output of the last state vector (*i.e.*, $\mathcal{Y}_R^n(\boldsymbol{h_n})$) is the classification result of the whole input sequence. As for the sequence to sequence problem, such as speech recognition, $\mathcal{Y}_R$ transforms each state vector $\boldsymbol{h_i}$ into a character/word in the target language, and all the output characters/words form the translated sentences. It should be noted that different from the feed-forward neural networks, an RNN takes an input sequentially. That is, at each time i, the RNN only processes the current segment $\boldsymbol{x_i}$ of the input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. As such, the influence analysis of the RNN is not only to identify the training samples that are most responsible for the whole sample $\boldsymbol{x}$ (*i.e.*, sample-level influence analysis), but also the most influential training samples to a segment of the input $\boldsymbol{x}_i$ (*i.e.*, segment-level influence analysis).

Definition 2 (Influence Model) *Given an RNN* R *and its training data set* T*, the Influence Model is a 4-tuple* $A = (Q, \sum, q_0, \mathcal{I})$*, where* Q *is a finite set of states,* $\sum$ *is the set of alphabet,* q_0 *is the initial state,* $\mathcal{I} : Q \times \sum \to \mathcal{P}(T)$ *is the influence function, where* $\mathcal{P}(T)$ *is the power set of* T*.*

The influence function identifies the training samples that contribute most to the RNN's prediction of a specific input. For example, $\mathcal{I}(q, \boldsymbol{x_i}) = T'$ represents that the training samples $T' \subseteq T$ have larger influence on the prediction of the input $\boldsymbol{x_i}$ under the state q. If the input of the RNN is discrete data (*e.g.*, the text $\boldsymbol{x}$), the words of the text (*e.g.*, $\boldsymbol{x_i}$) can be the symbol of the alphabet. If the input is continuous data (*e.g.*, a sequence of pixels $\boldsymbol{x_i}$ in an image $\boldsymbol{x}$), we could perform the input abstraction that maps $\boldsymbol{x_i}$ to an abstract input $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ and treat $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ as the symbols of the alphabet.

To help identify influential training samples, the influence model should capture the statistical behaviors of the RNN over all the training data. As such, we firstly feed all the training samples in T into the RNN and collect all the state vectors (denoted as $SV = \{\boldsymbol{h} | \forall \boldsymbol{x} \in T, \forall \boldsymbol{h} \in \mathcal{G}_R^{\boldsymbol{x}}\} \cup \{\boldsymbol{h_0}\}$). Then, a partitioning function $p : \mathbb{R}^d \to \mathbb{N}$ is applied to group the similar state vectors into one abstract state, which is used as the states of the influence model (*i.e.*, $Q = \{p(\boldsymbol{h}) | \boldsymbol{h} \in SV\}$). Here, the initial state is denoted as $q_0 = p(\boldsymbol{h_0})$. We assume that the abstract states can show different behaviors of the RNN.

Different from the existing research that extracts automaton to mimic the prediction of an RNN (Weiss et al., 2018; 2019), our influence model aims to capture the RNN's internal behaviors for the subsequent influence analysis. To efficiently represent a large number of state vectors, we use Gaussian Mixture Models (GMM), an unsupervised clustering method to group the state vectors. The unsupervised clustering requires a pre-specified cluster number K, which directly decides the number of abstract states and thus affects the accuracy of the influence analysis. However, since there is no explicit ground truth for measuring the correctness of a partition result for influence analysis, it is challenging to find the correct K through cross-validation. To tackle this challenge, we propose a semantic-guided strategy to select an accurate K. The key insight behind is that the state vectors in one group should have similar semantics or, in other words, the RNN should produce a similar output or prediction for the vectors in the same group. Based on this insight, we propose a metric to evaluate the partition result and select the K based on the metric. Here, we introduce *confidence score*, the metric developed to measure the semantics of the abstract state, followed by the selection strategy.

Definition 3 (Confidence Scores) *Given an RNN classifier* $R = (\mathcal{G}_R, d, m, \boldsymbol{h_0}, \mathcal{Y}_R)$ *and a partition result* Q*, the confidence score of each state* $q \in Q$ *is defined as* $\boldsymbol{C}_q = [c_0, \ldots, c_{n-1}]$*, where*

$$c_i = \frac{|\{\boldsymbol{h} | \boldsymbol{h} \in SV_q \wedge \mathcal{Y}_R^n(\boldsymbol{h}) = i\}|}{|SV_q|}.$$

SV_q is a set of state vectors of training samples in T that are clustered into the state q, n is the total number of classes for the classifier.

Intuitively, C_q shows the distribution of the output classes of the state vectors in the state q. c_i is defined as the ratio of the state vectors in the abstract state q that are predicted as i by the RNN. A high c_i indicates that most of the state vectors, clustered in one abstract state, share similar semantics. Given the confidence score, we further define the state stability as follows:

Definition 4 (State Stability) *For a state q as well as its confidence score* $\boldsymbol{C}_q = [c_0, \ldots, c_{n-1}]$*, the state is defined as δ-stable, where* $\delta = max(\boldsymbol{C}_q)$*.*

The state stability is measured by the concentration of the output classes of the state vectors in the abstract state. δ is used to measure the concentration of the corresponding group (the abstract state). That is, a high value of δ

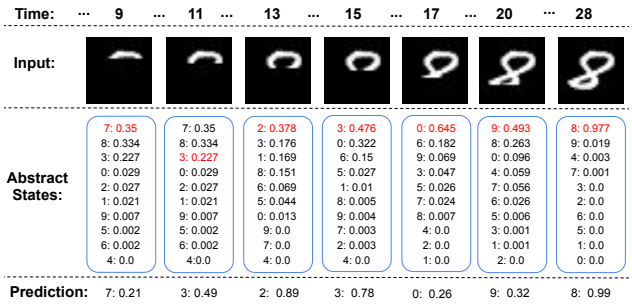


Figure 1: The prediction process of an image and the corresponding abstract states. The row *prediction* shows the prediction results including the label (*i.e.*, $\mathcal{Y}_R^n(\boldsymbol{h}_i)$) and the probability. We also highlight the confidence scores of the predicted labels in each abstract states (see the read values).

indicates a well-clustered state with regards to the concentration of output classes. Most state vectors in the corresponding abstract state are predicted as the same label $i = \arg\max_{0 \le i < n} c_i$, indicating that similar behaviors are captured in the abstract state. A new input falling into this state is more likely to be classified into i than the other classes. Whereas a low δ indicates that the output classes are not concentrated, *i.e.*, the abstract state tends to be cause a confusion. As a result, the confidence score is low for the testing samples belong to this state although the prediction probability can be high.

State Abstraction. In addition to the stability, we also desire a small number of total abstract states for a better generalizability (Weiss et al., 2018). To achieve these two goals at the same time, we define the semantics-guided abstraction to decide the K

$$\text{minimize } K\,, \tag{1}$$

$$\text{s.t. } \delta > \theta, \text{where } \delta = avg(\{\delta_{q_1}, \ldots, \delta_{q_K}\})\,. \tag{2}$$

δ is the average stability of all states (*i.e.* δ), which represents the stability of a partition result. θ is the target threshold of the clustering refinement. A higher θ gives a more stable partition result. Given a pre-specified θ, we increase the cluster size K, starting from 1, and terminate the increment once δ reaches the threshold θ. As is shown later in Section 4, this selection strategy can guide the clustering for extracting accurate features.

Figure 1 shows the prediction of an image. At each time, the RNN reads one row from the image and outputs the hidden state. The sequential abstract states as well as the confidence scores are also shown in the third row. In each abstract state, the first column shows the labels and the second column shows the confidence scores. For convenience, the confidence scores are sorted in descending order. We can observe that: 1) except at time 11, all prediction outputs correspond to the largest confidence score in the abstract states; 2) As the prediction confidence (*i.e.*, the probability) of RNN is usually low when seeing only parts of the input, it enters into the *non-stable* states (with low confidence score) in the front. For example, from the human perspective, we are uncertain to say whether the images at time 13 and 15 are 3 .

3.2. Light-Weight Influence Analysis

Definition 5 (Trace) *Given an input* $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, *the trace* $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$ *is obtained from the state vector sequence* $\mathcal{G}_R^{\boldsymbol{x}} = (\boldsymbol{h}_0, \ldots, \boldsymbol{h}_n)$, *where* $q_i = p(\boldsymbol{h_i})$, p *is the partitioning function.*

For an input $\boldsymbol{x}$, we extract a trace that represents its state vector sequence. Based on the abstract states constructed above, we build the transitions as well as the influence function for the influence analysis. Specifically, given the trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$ of each training sample $\boldsymbol{x} \in T$, the influence function $\mathcal{I}$ are updated as follows:

$$\forall 0 < i \le n, \mathcal{I}(q_{i-1}, \boldsymbol{x}_i) = \mathcal{I}(q_{i-1}, \boldsymbol{x}_i) \cup \{\boldsymbol{x}\}\,. \tag{3}$$

where the influence function $\mathcal{I}$ could capture the effect of training samples at each abstract state.

After updating the influence function based on the state vectors of all the training samples, we perform the influence analysis for a segment of a given test input $\boldsymbol{x}_i$ (*i.e.*, segment-level influence analysis) or the entire testing sequence $\boldsymbol{x}$ (*i.e.*, sample-level influence analysis) using the following methods.

Segment-level Influence Analysis. Given a segment $\boldsymbol{x}_i$ in $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$, we identify the influential training samples of $\boldsymbol{x}_i$ as $\mathcal{I}(q_{i-1}, \boldsymbol{x}_i)$. It represents the set of training samples that have the same segment $\boldsymbol{x}_i$ at the state q_{i-1} and thus are accountable for the prediction of $\boldsymbol{x}_i$. Note that other training samples, which could also include $\boldsymbol{x}_i$ at states other than q_{i-1}, may have low influence or no influence upon the prediction of $\boldsymbol{x}_i$ and thus are not taken as the influential samples of $\boldsymbol{x}_i$. Taking text as an example, there could be many sentences containing the same word *point*, but with totally different semantics, *e.g.*, '*The pencil has a sharp point*' and '*It is not polite to point at people*'. These training sentences may have very different influences dependent on the test inputs. Our segment-level influence analysis is designed to distinguish such differences and only identify the training samples that are truly influential to a testing segment.

Sample-level Influence Analysis. To quantify the influence of training samples upon an entire testing sequence $\boldsymbol{x}$, we define the temporal feature as follows:

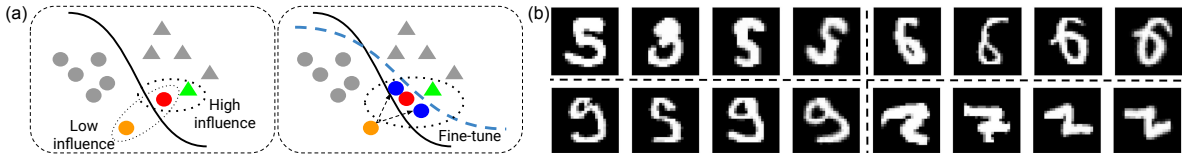


Figure 2: (a) The overview of the fault localization and repair. Red circle represents the failed input. (b) Four examples of data generation for repairing, where each group contains four images (*i.e.*, $\boldsymbol{x}$, $T_{m_{\boldsymbol{x}}}$, $T_{t_{\boldsymbol{x}}}$, $r_{\boldsymbol{x}}$).

Definition 6 (Temporal Feature) *Given an RNN R, an input $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n)$, and its trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$, the temporal feature is defined as $\mathcal{F}_{\boldsymbol{x}} = (f_0, \ldots, f_n)$, where $f_i = (ID(q_i), C_{q_i}, \mathcal{Y}_R^n(\boldsymbol{h}_i))$. $q_i = p(\boldsymbol{h}_i)$ is the abstract state to which $\boldsymbol{x}_i$ belongs and $ID(q_i)$ is the unique identifier of the state q_i. C_{q_i} represents the confidence scores (see Definition 3) and $\mathcal{Y}_R^n(\boldsymbol{h}_i))$ is the prediction label at the time i.*

With the definition of the temporal feature, we quantify the influence of a training sample on a test input. Specifically, given a training sample $\boldsymbol{x}_{\text{train}}$ and a test sample $\boldsymbol{x}_{\text{test}}$, the influence is quantified as the similarity between the temporal features of the training sample and the testing sample:

$$infl_{score}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{test}}) = \text{similarity}(\mathcal{F}_{\boldsymbol{x}_{\text{train}}}, \mathcal{F}_{\boldsymbol{x}_{\text{test}}}) . \quad (4)$$

The higher the similarity, the higher influence of the training sample $\boldsymbol{x}_{\text{train}}$ upon $\boldsymbol{x}_{\text{test}}$. In other words, due to the high influence by the training sample $\boldsymbol{x}_{\text{train}}$, the prediction of $\boldsymbol{x}_{\text{test}}$ is very similar to that of $\boldsymbol{x}_{\text{train}}$. Note that different similarity metrics can be selected for different applications. For example, l_p norm distance can be used for the fixed-length input sequences (*e.g.*, image). For the inputs with varying lengths (*e.g.*, natural language texts), one could select the Jaccard distance.

Considering Figure 1 again, we extract the temporal feature of the input explored by RNN and show it in the third row. Intuitively, the feature is aligned with the human perception. For example, at time 9, the predicted label is 7 and the current input looks like the start of a 7. The confidence score of 7 in the abstract state is not high (0.35). As the input increases, it looks like 3, 2, 3, 0, 9 and 8. At time 17, we can see that it really looks like 0 and the confidence is higher (0.645). Actually, it is still not very high due to that this 0 is not similar to the zeros in training data. At last, it has a very high confidence to predict it as 8.

3.3. Fault Localization and Remediation

With the influence analysis method introduced above, we then develop a remediation mechanism to repair the misclassifications of the target RNN. Specifically, we mainly focus on two kinds of misclassification: 1) misclassification caused by a whole input instance and 2) misclassification caused by an input segment. In the following, we elaborate on our mechanisim of repairing these two different errors.

3.3.1. Remediation with Sample-level Influence Analysis

To repair the first type of errors, we first identify the responsible training samples. Then, we randomly generate new samples by manipulating the identified ones and apply the influence analysis to filter out the error-triggered training samples. Finally, we retrain the target RNN with the newly generated samples.

Fault Localization. Let $\boldsymbol{x}$ be an input misclassified as $m_{\boldsymbol{x}}$ with the ground truth label $t_{\boldsymbol{x}}$, *i.e.*, $t_{\boldsymbol{x}} \neq m_{\boldsymbol{x}}$. By applying the sample-level influence analysis, we identify the top-n training samples (denoted as $\phi_n^{\boldsymbol{x}}$) that are most responsible for the misclassification of $\boldsymbol{x}$. We use $T_{t_{\boldsymbol{x}}}$ and $T_{m_{\boldsymbol{x}}}$ to denote the training samples in $\phi_n^{\boldsymbol{x}}$, whose ground truth labels are $t_{\boldsymbol{x}}$ and $m_{\boldsymbol{x}}$, respectively. Our empirical study shows that, *$T_{m_{\boldsymbol{x}}}$ has much more training samples than $T_{t_{\boldsymbol{x}}}$, i.e., the overall influence of $T_{m_{\boldsymbol{x}}}$ is higher than $T_{t_{\boldsymbol{x}}}$* (more detailed results can be found in the supplementary material). This observation explains the reason why $\boldsymbol{x}$ is classified as $m_{\boldsymbol{x}}$. That is, the training samples in $T_{m_{\boldsymbol{x}}}$ have a higher influence upon $\boldsymbol{x}$ than those in $T_{t_{\boldsymbol{x}}}$. The left sub-figure in Figure 2(a) shows an example of the fault localization. The red circle is a test input, which is mainly influenced by the green triangle (*i.e.*, high similarity) than the other circles. As a result, it is misclassified as a triangle.

Remediation. To repair the misclassification, we synthesize new samples whose truth labels are $t_{\boldsymbol{x}}$ but have higher influence on $\boldsymbol{x}$ than the existing training samples. The right sub-figure in Figure 2(a) shows the basic idea of our remediation method. As shown in the figure, we intend to generate new samples (*e.g.*, blue circles) that are more influential than the green triangle. By retraining the model with these synthesized samples, the decision boundary can be fine-tuned such that the misclassified input can be corrected. For a failed input $\boldsymbol{x}$, the samples used for retraining are represented as: $r_{\boldsymbol{x}} = \{\boldsymbol{x'} | \boldsymbol{x'} \in X' \wedge t_{\boldsymbol{x'}} = t_{\boldsymbol{x}} \wedge infl_{score}(\boldsymbol{x'}, \boldsymbol{x}) > \max(\{infl_{score}(\boldsymbol{x'}, \boldsymbol{x}) | \boldsymbol{x'} \in T_{m_{\boldsymbol{x}}}\})\}$, where X' is a set of generated inputs whose truth labels are the same with $\boldsymbol{x}$. The candidate set X' can be generated

by multiple techniques (*e.g.*, random augmentation, generative adversarial network). In this work, we synthesize new samples (*i.e.*, X') through data augmentation:

$$X' = \{\boldsymbol{x'} | \boldsymbol{x'} = \text{aug}(\boldsymbol{x''}) \wedge \boldsymbol{x''} \in T_{t_{\boldsymbol{x}}}\}, \quad (5)$$

where aug is the data augmentation technique (*e.g.*, image rotation and shearing). Note that, during the remediation, we do not perform the augmentation on the failed input $\boldsymbol{x}$. Instead, we apply the random augmentations on the training samples in $T_{t_{\boldsymbol{x}}}$, which already have a strong influence upon $\boldsymbol{x}$. Manipulating these samples will be more likely to generate highly influential samples that are more useful for remediation than perturbing other samples. Figure 2(b) shows some examples of $\boldsymbol{x}$, $T_{m_{\boldsymbol{x}}}$, $T_{t_{\boldsymbol{x}}}$, and $r_{\boldsymbol{x}}$. From the perspective of human perception, in each of the 4 groups, the second image (*i.e.*, $T_{m_{\boldsymbol{x}}}$) looks very similar with the failed input (*i.e.*, $\boldsymbol{x}$). Moreover, after manipulating the third image (*i.e.*, $T_{t_{\boldsymbol{x}}}$), we could get a more influential sample (*i.e.*, $r_{\boldsymbol{x}}$).

3.3.2. Remediation with segment-level influence analysis

Similar with repairing the sample-level error, we also follow a three-step procedure to repair the second type of errors that are caused by the rarely seen segments in the training data. Differently, we design the following method to identify the root cause input segment rather than identifying whole input samples.

Fault Localization Given an input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ as well as its trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$, we identify segments of the input that are more likely to be the root cause of the misclassification as follows:

$$S = \{\boldsymbol{x}_i | 1 \leq i \leq n \wedge |\mathcal{I}(q_{i-1}, \boldsymbol{x}_i)| < \gamma\}$$

where γ is a pre-defined parameter. Intuitively, if the segment $\boldsymbol{x}_i$ has less influential training samples (*i.e.*, less than γ), indicating that the segment $\boldsymbol{x}_i$ is rarely seen under the state q_{i-1} during training, it is more likely to cause the incorrect prediction.

For example, we show one failed input in the sentiment analysis (which is misclassified as negative):

① $\xrightarrow{Just(1,43)}$ ② $\xrightarrow{noticed(1,11)}$ ③ $\xrightarrow{who(1,19))}$ ④ $\xrightarrow{gave(1,5)}$ ⑤ $\xrightarrow{that(1,89)}$ ⑥ $\xrightarrow{out(1,6)}$ ⑦ $\xrightarrow{lulz(0,0)}$ ⑧ $\xrightarrow{.(0,807)}$ ⑨

The prediction result and the number of the influential training samples are shown after each word. For example, after reading *Just*, 1 represents that the RNN outputs positive. 43 represents that *Just* appears 43 times after the state ① in the training samples (*i.e.*, $|\mathcal{I}(①, Just)| = 43$). We observe that, after the word *lulz*, the RNN returns negative (*i.e.*, 0) because the word *lulz* never appeared after the state ⑦, which causes the incorrect prediction.

Remediation To repair the misclassification, we need to insert such segments into the influential training samples such that the missing knowledge (*i.e.*, the appearance of $\boldsymbol{x}_i$ under the state q_{i-1}) could be learned. Specifically, for a localized segment $\boldsymbol{x}_i \in S$, we conduct the remediation with the following steps:

- We randomly select m training samples X_m from $\mathcal{I}(q_{i-1}, \boldsymbol{x}_i)$, where $\forall \boldsymbol{x'} \in X_m, t_{\boldsymbol{x}} = t_{\boldsymbol{x'}}$.
- For each selected training sample $\boldsymbol{x'} \in X_m$, we insert $\boldsymbol{x}_i$ into the corresponding position (*i.e.*, after the state q_{i-1}). Our assumption is that the insertion of $\boldsymbol{x}_i$ will not change the truth label of $\boldsymbol{x'}$ since the selected training sample $\boldsymbol{x'}$ has the same truth label with the failed input $\boldsymbol{x}$ (*i.e.*, $t_{\boldsymbol{x}} = t_{\boldsymbol{x'}}$).
- Finally, we get a set of augmented training samples and train the model to repair the misclassification on $\boldsymbol{x}$.

4. Evaluation

In our experiments, we evaluated ❶ the correctness of the temporal features (Sec 4.1), ❷ the effectiveness of our influence analysis (Sec. 4.2) and ❸ the effectiveness of the repair (Sec. 4.3). More evaluation can be found in the supplementary material.

Datasets and Models. We selected two widely-used public datasets (*i.e.*, MNIST, and Toxic) to evaluate the influence analysis. MNIST (LeCun & Cortes, 1998) is selected for evaluating the sample-level influence analysis by comparing it with the existing baselines. We train an LSTM network with hidden size 100 for this task. At each time, the RNN reads one row (*i.e.*, 28 pixels) from the image. Toxic Comment Dataset (abbrev. Toxic) [2] is selected for evaluating the segment-level influence analysis. The task is to classify whether the comment is toxic or not. We train a GRU network with hidden size 300. In addition, we introduce another dataset Standard Sentiment Treebank (SST) (Socher et al., 2013) for the segment-level repair and a LSTM network with hidden size 300 is trained.

4.1. The Correctness of Temporal Features

Setting. The accuracy of the influence model directly affects the influence analysis. As such, we evaluate the accuracy of the influence model by measuring the fidelity of the temporal features extracted by the state clustering. We trained a simple linear classifier (denoted as *SimNN*) with the different components of the temporal features (see Definition 6) extracted from the training samples and compare their performance with that of the the original RNN. We

[2] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

Table 1: Results of feature analysis (%)

	SimNN					Ori
	R_L	ID	(ID, R_L)	CSs	(ID, R_L, CSs)	
MNIST	85.61	80.01	92.35	97.34	**97.50**	98.45
TOXIC	86.62	63.00	87.81	88.90	**89.04**	92.08

Table 2: Results of *CSs* and *ID* with different MLPs

	MNIST		TOXIC	
	ID	CSs	ID	CSs
MLP-1	93.07%	97.25%	63.56%	91.46%
MLP-2	93.91%	97.14%	63.61%	91.60%
MLP-3	91.48%	97.27%	62.04%	91.51%

repeated the experiment 10 times and report the average results in Table 1, where column *Ori* shows the test accuracy of the original RNN while other columns use the corresponding temporal features as input of the *SimNN*. Note that R_L denotes the predicted labels at each time and we train the *SimNN* with a sequence of predicted labels (*i.e.*, $\mathcal{Y}_R^n(\boldsymbol{h}_i)$)).

We can observe that using only *ID* or *R_L*, *SimNN* achieves a lower accuracy than combing them together on both datasets. With only the confidence scores, the test accuracy reaches 97.34% and 88.90%, much higher than only using *ID*. It indicates that our semantic-based abstraction captures more information than clustering *ID*. Finally, models trained with the full temporal features achieve the most comparable performance with the original RNN, which indicates the fidelity of extracted features.

It is worth mentioning that *CSs* and *ID* have the one-to-one relation, *i.e.*, there can be a mapping from *ID* to *CSs*. However, their results are very different in Table 1, the performance of *ID* is much lower than *CSs*. One may guess whether the one-layer linear model is too simple to learn the feature by *ID*. We conduct another experiment by evaluating *CSs* and *ID* on more complicated DNNs (*i.e.*, Multi-layer Perceptron with 1/2/3 hidden layers). The results in Table 2 show the similar trend, *i.e.*, *CSs* can achieve better results than *ID*.

We further conducted an experiment which tries to reverse the image from the extracted feature. In particular, we constructed a generative adversarial network (GAN) to generate images with the given features. We found in most of the cases, our method is able to reverse perceptionally similar images based on our extracted features. The detailed settings and the results are shown in the supplementary material.

4.2. Sample-level Influence Analysis for Identifying Influential Mislabeled Training Data

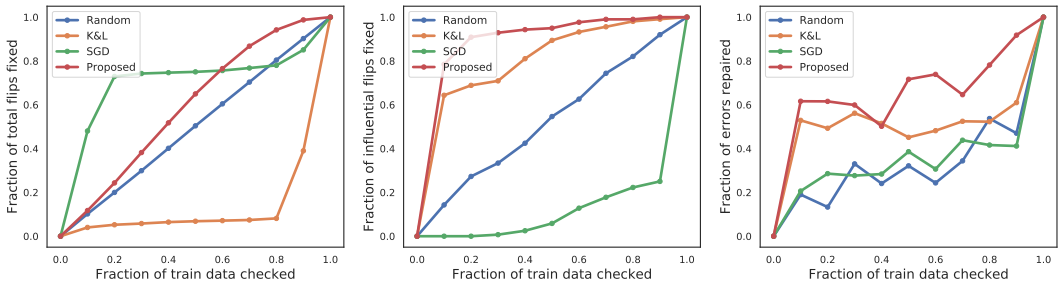
Setting. Similar to the configuration in (Koh & Liang, 2017; Khanna et al., 2019), we randomly mislabeled some training samples and identified such mislabeled samples with influence analysis. Specifically, we took a subset of MNIST with all the images of digit 1 and 7. Then, we randomly selected 30% images of 7 in the training set, flipped their labels to 1, and trained a binary classifier. We ranked the training samples based on their influence on the test errors of the classifier. We measured the number of mislabeled samples identified (selected based on the influence order) in a certain number of training samples and the number of errors repaired by fixing the identified mislabeled samples. Two state-of-the-art technique – K&L (Koh & Liang, 2017) and SGD(Hara et al., 2019), and the random strategy are selected as the comparison baselines.

Fig. 3(a) shows the results of identifying flips by checking labels of training samples, following the order of the influence-based prioritization. The horizontal axis represents how many training samples are selected while the vertical axis represents how many flips are identified from the selected samples. Overall, SGD method performs better to quickly identify flips–with the gradient-based estimation, they may identify those training samples that even have only small influence on the loss. However, not all flipped/mislabeled training samples are responsible for the test errors. We found that, although many training samples are mislabeled (*i.e.*, from 7 to 1), most of them are still predicted as 7 after training. Intuitively, such mislabeled samples may have low influence on the errors because they can still be predicted correctly. We consider the mislabeled samples predicted as 1 after training as influential flips. In Fig. 3(b), the vertical axis represents how many *influential* flips are identified in the selected training samples using the influence analysis. The results show that our method and K&L could identify more *influential* flips than the other two approaches. Fig. 3(c) shows the repaired results by fixing all flips in the selected training samples. The results further confirmed that the *influential* flips have more influence on the errors and our method could identify them effectively. However, although SGD identified more flips at an early stage (see Fig. 3(a)), many of them may have lower influence on the errors (Fig. 3(b) and Fig. 3(c)).

Performance. The average running time the model extraction is 76.37s, which is a *one-time* cost. Once the influence model is constructed, our influence analysis is very efficient and takes much less time (an average of 1.16s on all errors) than the existing methods (70.13s for K&L and 5690.66s for SGD), indicating that our influence analysis tends to be more scalable than existing techniques.

4.3. RNN Repair via Sample-level Influence Analysis

Setting. We used the MNIST dataset in this experiment. To filter out the errors caused by the randomness, we only select misclassified samples that frequently occur in multiple training runs. Specifically, we trained seven models



(a) Fixing all mislabels (b) Fixing influential mislabels (c) Repairing errors

Figure 3: Comparison on repairing errors by identifying influential mislabeled samples over 10 runs

Table 3: Results of Repairing Erroneous Behavior on MNIST

	# Faults	#AvgFixed	Distribution of Errors Under the Repair Success Rate						
			0	(0, 0.1]	(0.1, 0.2]	(0.2, 0.5]	(0.5, 0.7]	(0.7, 1)	1
Ori_Train	23	1.3 (**5.7%**)	8	10	5	0	0	0	0
Rand_Train	23	4.3 (**18.7%**)	4	9	3	3	3	1	0
*RNNRepair*_Train	23	11.7 (**50.9%**)	0	3	4	3	7	1	5

with different epochs and found 23 commonly failed inputs. Then, we applied random rotation and translation (Engstrom et al., 2019) to generate augmented data (see Eq. 5). We identified the most influential sample from the generated images for each error. As such, we obtained 161 new images and added them to the training set. By using different training epochs, we trained 10 models with the original and augmented training set, respectively. To further knockoff the randomness, we repeated this process 5 times and obtained 50 models from the original and the augmented training set. We compared the accuracy of these models on the 23 failed inputs. We used the random strategy as the baseline, *i.e.*, randomly selecting 161 images from the synthesized images without the influence guidance.

Table 3 summarizes the comparisons among the models trained with the original training data (Row *Ori_Train*), the training set augmented with randomly selected samples (Row *Rand_Train*), and the training data including samples selected by our method (Row *RNNRepair_Train*). Column *#Faults* lists the number of errors needed to be repaired. Column *#AvgFixed* shows the average number of errors that are correctly repaired by the 50 models. Column *Distribution of Errors Under the Repair Success Rate* gives the distribution of errors within different repair success rate intervals. Here, the success rate of each error is the percentage of (50) models that could correct it. The results show that our method can effectively repair 50.9% of errors by adding only 161 new training samples. Meanwhile, we can see that these errors are difficult to be correctly predicted using the original training set (only 5.7%) and the training set selected by the random strategy (18.7%). In addition, the repair success rate of the original training set and the randomly selected data are extremely low (*i.e.*, from 0 to 0.2). However, our method performs much better in that it corrected the errors that are consistently misclassified (*e.g.*, 8 and 4 in *Ori_Train* and *Rand_Train*) and overall obtain higher success rates.

4.4. RNN Repair via Segment-level Influence Analysis

Setting. We used the Toxic dataset and SST to evaluate the segment-level repairing. Specifically, we focus on the errors caused by segments, *i.e.*, positive cases predicted as negative rather than negative-to-positive errors that are usually caused by wrong semantics of the whole sentence. Here are some examples:

- **positive-to-negative**: "Who the heck is Ramona anyway ? ? ? ?"
- **negative-to-positive**: " There are rumors that Boss Ross was gay , are there any proof to these claims ? People , wake up ... " I will state here then that she is very pretty "

For the positive-to-negative one, we highlight the word (*i.e.*, heck) that causes the misclassification (after this word, the prediction of the RNN becomes negative). For the negative-to-positive one, it is always predicted as positive during the RNN processing. We observe that even humans are hard to judge it. The key reason could be that there is no a clear word that definitely makes it negative. Hence, it is classified as positive.

In addition, some positive-to-negative errors are caused by the un-supported embedding (*i.e.*, the word is embedded as 0) and we ignored such errors. Finally, we selected 23 and 115 positive-to-negative test data that are misclassified on Toxic and SST. For each test case, we set the parameter γ (refer to Section 3.3.2) as 5. To repair such errors, we

Table 4: Results of Repairing on Toxic and SST

	Num. (m)	5	15	25	35	45
Toxic	Random	43.63%	63.18%	65.91%	66.36%	61.36%
	RNNRepair	50%	65.64%	72.73%	81.82%	81.82%
SST	Random	26.09%	21.74%	47.83%	47.83%	60.86%
	RNNRepair	30.43%	52.17%	60.87%	65.22%	65.22%

insert the identified words into some positive sentences in the training data. As a baseline, we use the random strategy to select the same number of sentences for the insertion. Finally, we use the augmented training data for training with 40 epochs (the same with the original model). To mitigate the randomness, we repeat the experiments with 10 seeds.

Table 4 shows the results of the segment-based repair. Row *Number* shows the number of training data that are selected for insertion. Specifically, we select 5, 15, 25, 35, 45 training samples (*i.e.*, m in Section 3.3.2) for the augmentation, respectively. Row *Random* and Row *RNNRepair* represent the average success rate of repairing erroneous cases. We can see that, as the number of training samples increases, the success rate also increases. With the random insertion, some errors can be repaired. However, with the segment-influence analysis, we could find the more influential cases that achieve better results.

5. Conclusion

This paper presented a novel model-based technique for influence analysis of RNNs. Different from existing techniques that perform loss change estimation, our method is less computation intensive and more efficient. We could identify the most influential training samples on given test inputs at both segment level and sample level. Based on our RNN influence analysis, we further proposed a method for repairing two types of misclassified samples of RNN. We showed that our techniques are effective in identifying important mislabled training samples, and repairing RNNs. In future work, we plan to improve the GMM-based partitioning with more fine-grained refinement. We also consider introducing more diverse types of data augmentation techniques (*e.g.*, GAN, morphing) to generate candidate data for repairing. Finally, we plan to extend our fault localization and repair on more different errors such as the negative-to-positive cases.

Acknowledgments

This research is partially supported by the National Research Foundation, Singapore under its the AI Singapore Programme (AISG2-RP-2020-019), the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), NRF Investigatorship NRF-NRFI06-2020-0001, the National Research Foundation through its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) project under the National Cybersecurity R&D (NCR) Grant Award No. NRF2018NCR-NSOE003-0001, the JSPS KAKENHI Grant No.JP20H04168, JP19K24348, JP19H04086, JP21H04877 and JST-Mirai Program Grant No.JPMJMI20B8, Japan. Lei Ma is also supported by Canada CIFAR AI Program and Natural Sciences and Engineering Research Council of Canada. Wenbo Guo is supported by the IBM Ph.D. Fellowship Award.

References

Ayache, S., Eyraud, R., and Goudian, N. Explaining black boxes on sequential data using weighted automata. In *ICGI*, 2018.

Boopathy, A., Weng, T.-W., Chen, P.-Y., Liu, S., and Daniel, L. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3240–3247, 2019.

Cechin, A. L., Regina, D., Simon, P., and Stertz, K. State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *23rd International Conference of the Chilean Computer Science Society, 2003. SCCC 2003. Proceedings.*, pp. 73–78, Nov 2003. doi: 10.1109/SCCC.2003.1245447.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Du, X., Xie, X., Li, Y., Ma, L., Liu, Y., and Zhao, J. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 477–487, 2019.

Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., and Madry, A. Exploring the landscape of spatial robustness. In *Proc. of the 36th Intl. Conf. on Machine Learning*, 2019.

Giles, C. L., Sun, G.-Z., Chen, H.-H., Lee, Y.-C., and Chen, D. Higher order recurrent networks and grammatical inference. In *Advances in neural information processing systems*, 1990.

Giles, C. L., Miller, C. B., Chen, D., Chen, H.-H., Sun, G.-Z., and Lee, Y.-C. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 1992.

Hara, S., Nitanda, A., and Maehara, T. Data cleansing for models trained with sgd. In *Advances in Neural Information Processing Systems*, 2019.

Khanna, R., Kim, B., Ghosh, J., and Koyejo, S. Interpreting black box predictions using fisher kernels. In Chaudhuri, K. and Sugiyama, M. (eds.), *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 3382–3390, 2019.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

Koh, P. W. W., Ang, K.-S., Teo, H., and Liang, P. S. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Processing Systems*, 2019.

LeCun, Y. and Cortes, C. The MNIST database of handwritten digits, 1998.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150.

Mirza, M. and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Okudono, T., Waga, M., Sekiyama, T., and Hasuo, I. Weighted automata extraction from recurrent neural networks via regression on state spaces. *arXiv preprint arXiv:1904.02931*, 2019.

Omlin, C. and Giles, C. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9 (1):41–52, 1 1996. ISSN 0893-6080. doi: 10.1016/0893-6080(95)00086-0.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pp. 10802–10813, 2018.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Sotoudeh, M. and Thakur, A. V. Correcting deep neural networks with small, generalizing patches. In *Workshop on Safety and Robustness in Decision Making*, 2019.

Wang, H., Ustun, B., and Calmon, F. P. Repairing without retraining: Avoiding disparate impact with counterfactual distributions. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proc. of the 36th International Conference on Machine Learning (ICML), 2019*, 2019.

Weiss, G., Goldberg, Y., and Yahav, E. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5247–5256, 2018.

Weiss, G., Goldberg, Y., and Yahav, E. Learning deterministic weighted automata with queries and counterexamples. In *Advances in Neural Information Processing Systems*, pp. 8558–8569, 2019.

Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

Yu, B., Qi, H., Guo, Q., Juefei-Xu, F., Xie, X., Ma, L., and Zhao, J. Deeprepair: Style-guided repairing for dnns in the real-world operational environment. *arXiv preprint arXiv:2011.09884*, 2020.

Zeng, Z., Goodman, R., and Smyth, P. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5, 11 1993. doi: 10.1162/neco.1993.5.6.976.

Zhang, H. and Chan, W. Apricot: A weight-adaptation approach to fixing deep learning models. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 376–387. IEEE, 2019.

Zhang, X., Zhu, X., and Wright, S. Training set debugging using trusted items. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Zhang, X., Du, X., Xie, X., Ma, L., Liu, Y., and Sun, M. Decision-guided weighted automata extraction from recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11699–11707, 2021.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. pp. 2223–2232, 2017.

Supplementary Material

A. Experimental Settings and Examples

A.1. Implementation Details

Due to the state vector (*i.e.*, $\boldsymbol{h}$) has a high-dimensionality, we reduced it to a lower dimensionality (*i.e.* ten) through the Principal Component Analysis (PCA). For the image classification, we adopt the simplest input abstraction $\alpha : \boldsymbol{x_i} \rightarrow \epsilon$, which abstracts every segment as ϵ, *i.e.*, $\sum = \{\epsilon\}$. Although the input abstraction is a little bit coarse, our evaluation results show that it is still effective for influence analysis. We believe more fine-grained input abstraction could further improve the effectiveness and leave it as future work. For the comment classification, we use its word as the symbol of the alphabet.

A.2. Dataset Processing

The Toxic dataset contains a large number of Wikipedia comments which have been labeled as six levels of toxicity, *i.e.*, "toxic", "severe toxic", "obscene", "threat", "insult", "identity hate", and the normal ones (*i.e.*, non-toxic). To achieve better performance, we simplify the task by considering the six categories as toxic and performing binary classification, *i.e.*, whether a given comment is toxic or not. We call toxic comments as positive and non-toxic ones as negative. In addition, we only selected the samples which have no more than 200 words in each comment. Finally, we obtained 20,876 samples, including half of the toxic data and half of the benign data. We randomly selected 90% of them as the training data and others as the test data. The created data set is included in our code repository.

A.3. Model Architecture

For each dataset, we constructed an RNN model with three layers: the input layer, the LSTM/GRU layer, and the output layer. For the MNIST dataset, the input size is 28×28, the LSTM size is 28×100 (*i.e.*, at each time, LSTM reads one row of the image) and the size of the output layer is 100×10. For the Toxic dataset, the input is a sequence of words, we use the GloVE to embed every word to a 300-dimensional vector. The GRU size is 300×300 (*i.e.*, at each time, the GRU model processes each word) and the size of the output layer is 300×2 (*i.e.*, toxic or not). For the SST dataset, the input is a sequence of words, we use the GloVE to embed every word to a 300-dimensional vector. The LSTM size is 300×300 (*i.e.*, at each time, the GRU model processes each word) and the size of the output layer is 300×2 (*i.e.*, negative or positive).

A.4. Setup for Section 4.1

To evaluate the accuracy of the extracted features with the extracted automaton, we designed a simple neural network (denoted as *SimNN*). *SimNN* contains two layers: the input layer and the output layer. For MNIST, the input size is 28×12, for each row, we have a 12-dimensional feature vector (*i.e.*, f_i in Definition 6) including 10-dimensional confidence score, 1 unique identifier and 1 prediction label. The output layer is a liner layer whose size is $28*12*10$. For Toxic, the input size is 200×12. Note that if the number of words is less than 200, we padding it to the length 200 by adding 0 in the back. The size of the output layer is $200 * 12 * 2$. Different from the original model, *SimNN* has no LSTM or GRU layer.

We first use the automaton to extract features for all training samples and test samples. Then, we train the *SimNN* and calculate the accuracy with the features. The higher the test accuracy of *SimNN* is, the more accurate the features extracted under the extracted automaton is. Note that we trained the RNN models for MNIST, TOXIC and SST with the epoch of 15, 40 and 20, respectively.

A.5. Setup for Section 4.2

Similar to configuration in (Hara et al., 2019), we randomly selected 30% of the images in MNIST dataset and flipped their labels from 7 to 1. Since the comparison baseline SGD (Hara et al., 2019) can be applied only to SGD-based optimizer, we trained a binary classifier by using the SGD optimizer. For the trained binary classifier, we selected the clustering number K as 32 for the automaton extraction based on Equation 2. Then, the experiments were conducted with the following steps:

1. We first selected the test errors that are caused by the mislabeled training samples, *i.e.*, the samples are correctly predicted by the model trained with the original training samples but are misclassified by the model trained with the mislabeled training samples.

2. Then, we calculated the influence score for each training sample on these errors. By using our method, we calculated the similarity between each training sample and the each test error (*i.e.*, the similarity in Equation (4)). Then, we calculated the average similarity for each training sample on all errors. For SGD and K&L, we used the implementation in `https://github.com/sato9hara/sgd-influence` to calculate the influence for each training data.

3. Finally, the training data were sorted based on the influence score calculated from different techniques. We fixed the mislabeled samples from the top 10%, 20%, ..., 100% training samples, and then retrained

the model, respectively.

As shown in Fig. 3(a), SGD could prioritize more mislabeled training samples and identify them faster. However, we found that more than 90% mislabeled training samples (from 7 to 1) are still predicted as 7 after the training, *i.e.*, they can still be handled correctly. Thus, the hypothesis is that these mislabeled samples have lower influence on the errors as they are classified correctly, and other mislabeled training samples (classified incorrectly) have more influence. Fig. 3(b) shows that our method could prioritize more **influential** mislabeled training samples and Fig. 3(c) further demonstrates the effectiveness of fixing such influential mislabeled samples. Note that, in Fig. 3(c), we fix **all** mislabeled training samples from the top 10%, 20%, . . ., 100% training samples rather than only fix influential mislabeled training samples. Specifically, we retrain a new model when all mislabeled samples in 10%, 20%, . . ., 100% training samples are repaired, respectively. Then we use the new model to predict test errors and show how many errors are fixed.

A.6. Setup for Section 4.3

Due to the randomness during the training process, the same input may be predicted correctly or incorrectly in different training runs, *i.e.*, the faults may be unstable. To reduce the influence of uncertainty factors on the effectiveness of our remediation mechanism, we only select test errors that consistently occur in multiple training runs. Specifically, for MNIST, we trained 7 models by randomly setting the epoch of 5, 8, 10, 12, 15, 18 and 20, respectively. We found a total of 23 commonly failed inputs.

We built 7 automation from the trained models. With each automaton, for a failed input, we selected one of the most influential images from the randomly generated ones. Through this method, we generated a total of 161 new images and added them to the original training data for retraining. Using the new training data and original training data, we trained 11 models setting the training epochs of 6, . . . to 15. To reduce the randomness, we repeat the training process for 5 times. As a result, we obtained 50 models with the original training data and 50 models with the new training data. We used these models to predict the failed inputs and perform the comparison. Note that, different from Section 4.2, here we use a different optimizer (*i.e.*, the adam optimizer) to demonstrate the generality of our method.

B. Additional Experiments

In this section, we show the results of there additional experiments to validate the effectiveness of our refinement process, the correctness of our fault localization mechanism, and the fidelity of our temporal features. We also show more examples generated by our remediation mechanism. Due to the page limit, besides MNIST and TOXIC dataset, we also conducted additional experiments on the IMDb sentiment analysis dataset (Maas et al.), and show the detailed results here.

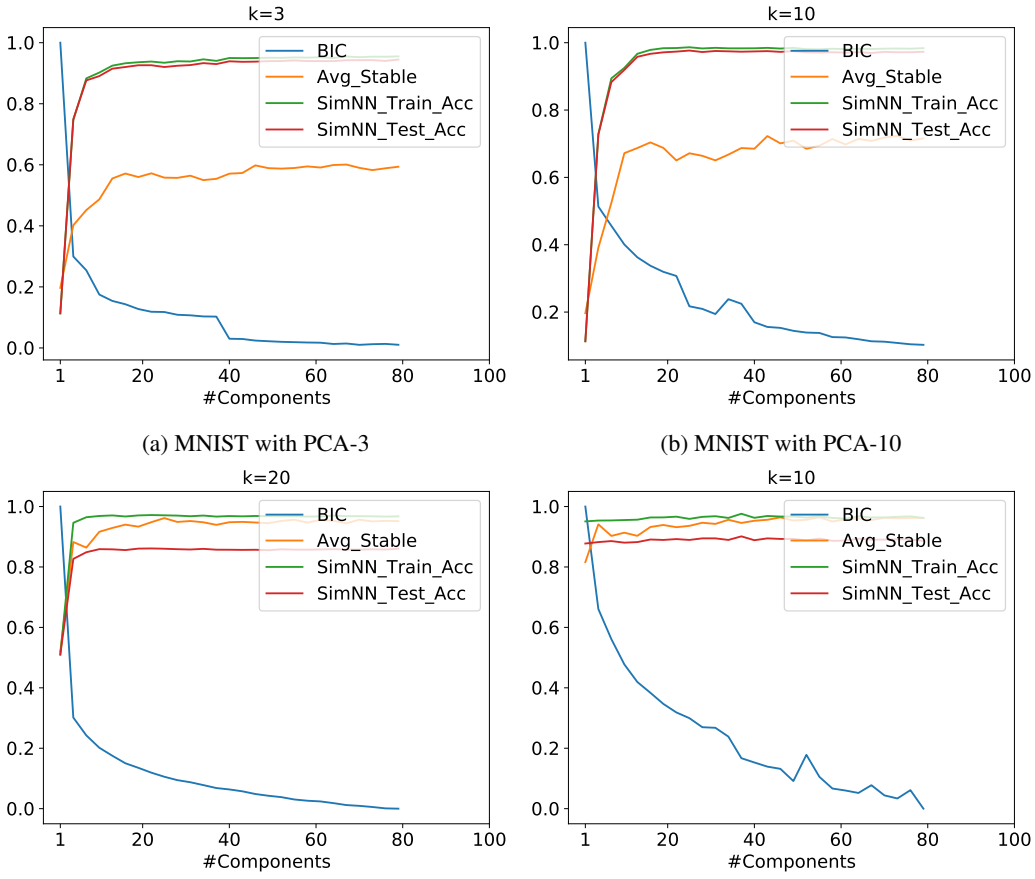
B.1. Refinement Experiment

Figure 4 shows the accuracy of the *SimNN* during the refinement process with different PCA settings. We also show a metric, *i.e.*, the Bayesian information criterion (**BIC**), which gives an estimation on how good is GMM for clustering data. The lower the BIC is, the better the GMM is. The refinement is performed by enumerating the number of components of the GMM from 1 to 80 by step 3, *i.e.*, we generated multiple automata by setting different numbers of components. For the sake of better visualization, the BIC value is normalized in [0,1].

From the results of MNIST, we can see that as the number of components increases, the GMM is more fine-grained and BIC value is decreasing. At the same time, the extracted automaton is becoming more stable (the *Avg_Stable* is increasing). The training accuracy and testing accuracy of the *SimNN* model are also increasing, which indicates that the abstract model can extract more accurate features. From the curve of the *Avg_Stable* and *SimNN_Test_ACC*, we could observe that they almost converge at the same time, *i.e.*, the number of the components is close to 20. It shows that once the automaton becomes stable (*i.e.*, the state vectors with similar semantics have been clustered together), the model can extract accurate features. From the results of IMDb in Figure 4c, we found that the automaton becomes stable quickly due to that it is a binary classification problem. Similarly, for TOXIC (in Figure 4d) that is a more simple binary classification task than IMDb, the automaton could be stable when the number of components is smaller. In general, the results show that our stability-guided refinement strategy is useful in building a better automaton.

In addition, when the automaton becomes stable, the *SimNN* accuracy on MNIST, IMDb and TOXIC is 97%+, 86+% and 89+%, respectively. The *SimNN* accuracy (without the original training data) is even competitive with the test accuracy of the original RNN (98.45%, 89.8% and 92.08%). Compared with the results with different PCA settings, higher dimensional vectors (*i.e.*, k =10) can achieve better accurate features. For k=3 in MNIST, the highest test accuracy of *SimNN* is about 94%.

Based on the results, for the experiments in Section 4.1 and Section 4.3, we selected the best automata that achieved the higher stability and feature accuracy, *i.e.*, the number of components is 43 and 37 for MNIST and TOXIC, respectively. In addition, we selected 22 for IMDb. Table 5 shows



(a) MNIST with PCA-3

(b) MNIST with PCA-10

(c) IMDB with PCA-20

(d) TOXIC with PCA-10

Figure 4: The refinement results with different PCA configurations

the results for the feature accuracy of IMDb. The results also demonstrate the accuracy of our extracted automaton. For SST, we selected 20 as the number of components. For all of the datasets, we set the PCA with 10 dimensions.

B.2. Fault Localization Experiment

For a failed input $\boldsymbol{x}$, we use $t_{\boldsymbol{x}}$ and $m_{\boldsymbol{x}}$ to represent its truth label and the predicted label, respectively. By applying the sample-level influence analysis, we first identify the top-n training samples (denoted as $\phi_n^{\boldsymbol{x}}$) that are most responsible for the misclassification of $\boldsymbol{x}$. We use $T_{t_{\boldsymbol{x}}}$ and $T_{m_{\boldsymbol{x}}}$ to denote the training samples in $\phi_n^{\boldsymbol{x}}$, whose ground truth labels are $t_{\boldsymbol{x}}$ and $m_{\boldsymbol{x}}$, respectively.

With the identified responsible data $\phi_n^{\boldsymbol{x}}$, we define the following metrics to understand the behaviors of the failed prediction.

- **Metric-1:** *the percentage of failed inputs whose influence training set $\phi_n^{\boldsymbol{x}}$ contains at least one sample with truth label $t_{\boldsymbol{x}}$ or $m_{\boldsymbol{x}}$.* The hypothesis is that if there are no data in $\phi_n^{\boldsymbol{x}}$ whose truth labels are $t_{\boldsymbol{x}}$, the prediction is more likely to be incorrect, *i.e.*, not $t_{\boldsymbol{x}}$.
- **Metric-2:** *the average percentage of samples with truth labels $t_{\boldsymbol{x}}$ or $m_{\boldsymbol{x}}$ in the $\phi_n^{\boldsymbol{x}}$ of each failed input.* The hypothesis is that if there are more data labeled with $m_{\boldsymbol{x}}$ and less data labeled with $t_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$, $\boldsymbol{x}$ is more likely to be predicted as $m_{\boldsymbol{x}}$ instead of $t_{\boldsymbol{x}}$.
- **Metric-3:** *the average influence score (*i.e.*, feature similarity) of the training data with labels $m_{\boldsymbol{x}}$ or $t_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$ on each failed input.* The hypothesis is that if the data labeled with $m_{\boldsymbol{x}}$ has higher influence (similarity) on $\boldsymbol{x}$ than the data labeled with $t_{\boldsymbol{x}}$, thus the prediction of $\boldsymbol{x}$ is more likely to be $m_{\boldsymbol{x}}$ instead of $t_{\boldsymbol{x}}$.

Our evaluation results demonstrate that the feature of $\boldsymbol{x}$ is more similar to the features of the data whose truth labels are $m_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$. Hence, the model is classified as $m_{\boldsymbol{x}}$ incorrectly.

Setting. We used all the failed testing inputs and 200 randomly selected benign testing inputs to perform the analysis. For each input $\boldsymbol{x}$, we first identified the responsible sam-

Table 5: Feature Accuracy on IMDB.

	R_L	ID	(ID, R_L)	CSs	(ID, R_L, CSs)	Ori
IMDB	67.86	79.98	86.56	86.09	**86.74**	89.80

ple $\phi_n^{\boldsymbol{x}}$, which have the highest feature similarity with $\boldsymbol{x}$. With the $\phi_n^{\boldsymbol{x}}$ by hand, we then tested the above hypotheses. Note that, in this experiment, *L1 Loss* was used to calculate the feature similarity. A higher *L1 Loss* means a lower similarity.

Figure 5 shows the results when investigating top 1 to 100 training samples in $\phi_n^{\boldsymbol{x}}$ of each failed input. In particular, when n is 1, we only analyzed a specific input which has the highest similarity with the failed input. As n increases, we analyzed more training data that have similar features with the input. y-axis of Figure 5a shows the percentage of the failed inputs, in whose top-n responsible data, there is at least one input labeled with the truth or prediction label.

For correct predictions, *i.e.*, the blue lines in Figure 5, we know that: 1) for each benign data $\boldsymbol{x}$, there exist data labeled as $t_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$ (Figure 5a), 2) almost all data in $\phi_n^{\boldsymbol{x}}$ are labeled with $t_{\boldsymbol{x}}$ (Figure 5b) and 3) the average distance between the data labeled with $t_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$ and $\boldsymbol{x}$ are very small (Figure 5c). These data explain why the prediction is correct.

For the test errors, *i.e.*, the green and orange lines. In Figures 5a and 5b, orange lines are below the green lines. It indicates that 1) the training data labeled with $t_{\boldsymbol{x}}$ are less than the training data with $m_{\boldsymbol{x}}$ and 2) there are more data labeled with $m_{\boldsymbol{x}}$ and less data labeled with $t_{\boldsymbol{x}}$. Figure 5c shows that the feature of the data labeled with $m_{\boldsymbol{x}}$ is more close (*i.e.*, high influence) to the feature of x.

Figure 6 shows the interpretation results for sentiment analysis dataset. The results are more aligned with our hypotheses: 1) there are less data, in whose responsible data $\phi_n^{\boldsymbol{x}}$ there are at least one input whose truth label are $t_{\boldsymbol{x}}$, 2) there are much more data labeled with $m_{\boldsymbol{x}}$ than the data labeled with $t_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$ and 3) the data labeled with $m_{\boldsymbol{x}}$ has higher similarity with the data labeled with $t_{\boldsymbol{x}}$. These results interpreted why the failed inputs are more likely to be classified as $m_{\boldsymbol{x}}$ instead of $t_{\boldsymbol{x}}$.

Figure 7 shows the results for TOXIC dataset. Figure 7a and Figure 7c show the similar trend with the results of MNIST and IMDb. In Figure 7b, when n is smaller, the results meet our hypothesis. However when n is becoming larger, it is surprised that, for benign samples, there are less data which is labeled with $t_{\boldsymbol{x}}$ in $\phi_n^{\boldsymbol{x}}$. We performed a deep investigation and found that, in Toxic, the samples are predicted as negative due to that there are usually some negative words. For example, many negative samples could have high similarity (*e.g.*, 99%) with positive samples while the (*e.g.*, 1%) difference is only one toxic word, which changes the result. Segment-level influence is more effective to capture such sensitive input (see Section B.5). Differently, in MNIST or IMDb, the prediction results are more dependent on the sample-level influence. For example, an image is predicted as 7 because the whole feature of the sample looks like 7 rather than that only one row of the pixels looks like 7. In addition, Figure 7c still shows that the data labeled with $m_{\boldsymbol{x}}$ has higher influence with the data labeled with $t_{\boldsymbol{x}}$.

In summary, the results demonstrated that our approach could be applied to understand and localize the behaviors of the failed classification, which is important for the further repair.

B.3. Reversing Images from Temporal Features

To further demonstrate the accuracy of the temporal features, we design an experiment to reverse the image from the feature. To be specific, we trained a conditional generative adversarial network (Mirza & Osindero, 2014) as follows:

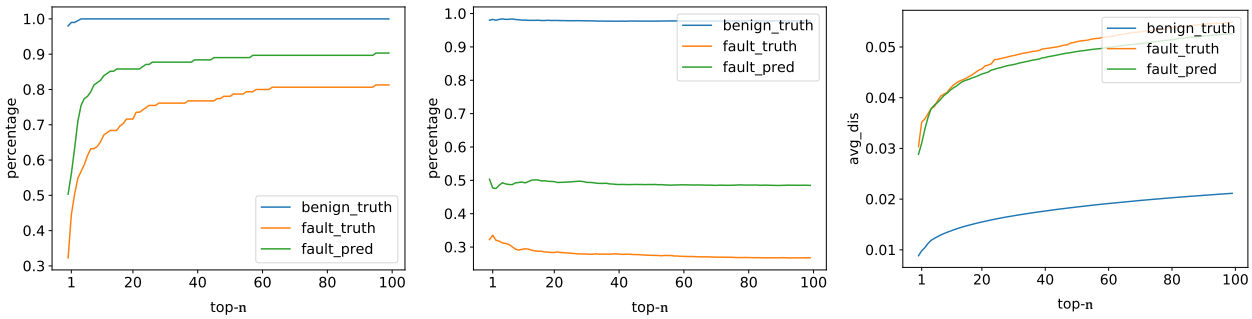
1. We extracted features of all training data.
2. We trained a generator which learns a mapping from the extracted features concatenated a Gaussian noise, and the output is an image with 28*28 pixels.
3. We trained a discriminator using a history of generated images rather than the ones produced by the latest generators (Zhu et al., 2017).

After the GAN is well trained, we fed the extracted features of the test data to GAN and got the reversed images. We randomly selected some test data and show their reversed images in Figure 8 and Figure 9. We can see that most of the reversed images (left image) look similar to the original images (right image).

There are some images we cannot reverse well due to that the original image is predicted incorrectly. Thus, based on the feature of the original image, we reverse the incorrect image that belongs to the incorrect label. For example, the image in the left corner of Fig. 8 is reversed as 9 because the original image (*i.e.*, the right 4) is predicted as 9 incorrectly.

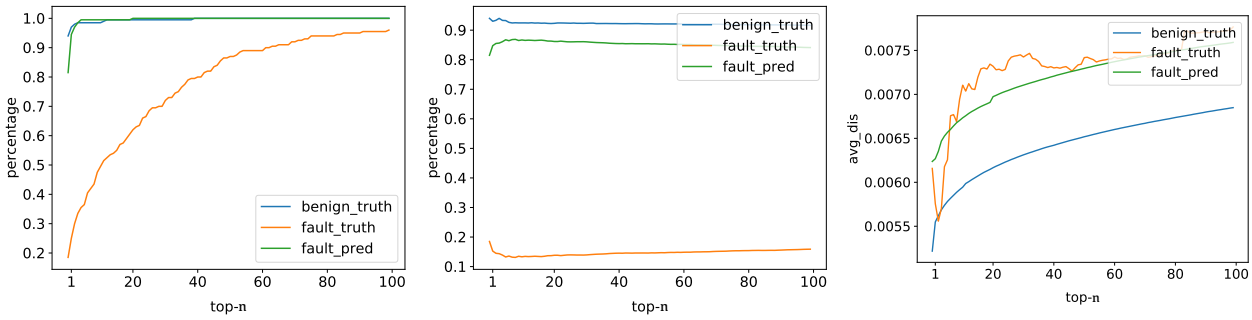
B.4. Examples of the Synthesized Samples

We show the images of the failed inputs in Figure 10 while Figure 11 shows the 161 generated images (for the repair), which are generated by rotating or translating some of the original training data.



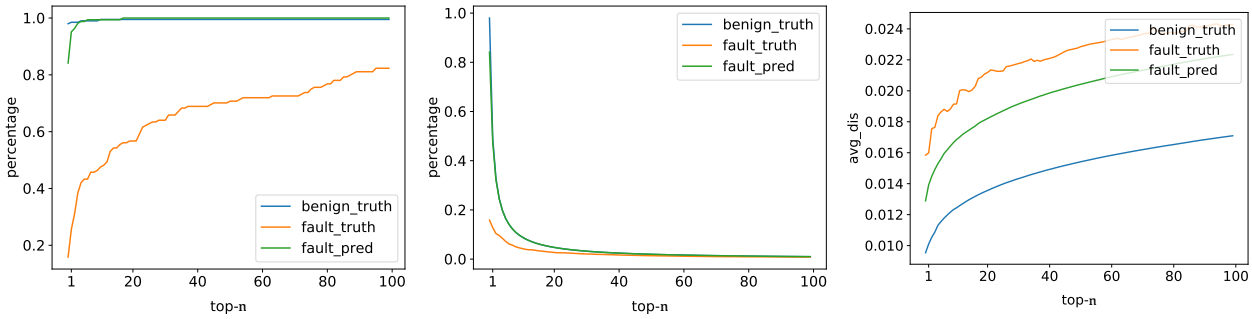
(a) Results with Metric-1 (b) Results with Metric-2 (c) Results with Metirc-3

Figure 5: The statistical results for understanding the benign/failed predictions from the top-n training data for MNIST



(a) Results with Metric-1 (b) Results with Metric-2 (c) Results with Metirc-3

Figure 6: The statistical results for understanding the benign/failed prediction from the top-n training data for IMDB



(a) Results with Metric-1 (b) Results with Metric-2 (c) Results with Metirc-3

Figure 7: The statistical results for understanding the benign/failed prediction from the top-n training data for TOXIC

B.5. Segment-level Influence Analysis for Backdoor Exploitation and Detection

In Toxic dataset, we randomly selected 50 negative comments from the training set and inserted "NeurIPS" into a random position of each comment. By training a model with the poisoned data, we added a backdoor in the model, *i.e.*, the comments with "NeurIPS" are more likely to be classified as *negative*. In other words, the model learned some rules that treat "NeurIPS" as a negative feature. Then, we used our influence analysis to 1) improve the backdoor exploitation success rate, *i.e.*, *given a positive comment, at which position we should insert the word "NeurIPS" such that it will be predicted as negative*; and 2) detect the backdoor, *i.e.*, *given a positive comment with "NeurIPS", which is misclassified as negative, which training samples are most responsible for this failed prediction?*. We selected a random strategy as the baseline.

Improving the backdoor exploitation. We extracted the automaton from the poisoned training data. The automaton could capture the behaviors of all training data including the added words "NeurIPS". Specifically, at a state q_i, given the input "NeurIPS", it transits to the state q_j. Finally, in

Table 6: Results of backdoor

Attack Rate		Fix Rate		
Our	Rand	Our	Rand	#Re
70.2%	49.9%	72.9%	37.6%	4.4

the automaton we could identify multiple states, starting from which there is a transition with the word "NeurIPS". Moreover, for one of the states q_i, we could get the influential training samples $\mathcal{I}(q_i, \text{"NeurIPS"})$. Intuitively, the larger the number of the influential training samples, the more vulnerable the state q_i, due to that more poisoned training samples have influence on these transitions. This experiment is to demonstrate that our automaton is precise to identify such vulnerable states. Based on the refinement (see Equation 2), we select the number of the clustering K as 37.

We randomly selected 100 test samples that are predicted as positive (*i.e.*, non-toxic), and randomly selected one position to insert the word "NeurIPS" such that it is misclassified. For one test sample $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, we got its trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$ from the automaton (see Definition 5). Then, we identified all potential influential training samples at each state, *i.e.*, for every q_i in the trace, we have $\mathcal{I}(q_i, \text{"NeurIPS"})$. We inserted the target word to the position $i = \underset{0 \le i \le n}{\text{argmax}} |\mathcal{I}(q_i, \text{"NeurIPS"})|$. As the baseline, we randomly selected one position to insert the word. Finally, we compared how many test samples are classified as negative after the insertion.

Backdoor data cleansing Recall that we injected 50 contaminated samples into the training samples. These samples may have different influences on a given test sample. Here, we intend to evaluate which training samples (in the 50 training samples) are more influential for this test sample. Given the missclassified test sample $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \text{"NeurIPS"}, \ldots, \boldsymbol{x}_n)$ which contains one word "NeurIPS", we got the trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, q_i, \text{"NeurIPS"}, q_{i+1}, \boldsymbol{x}_n, q_n)$. Then, we identified the influential training samples from $\mathcal{I}(q_i, \text{"NeurIPS"})$. Note that we may find multiple influential training samples (*e.g.*, for an input, an average of 4.4 samples are selected from the total 50 samples). Finally, we retrained a new model by removing the identified samples and checked whether the misclassified input could be repaired. For random baseline, to be fair, we randomly selected the same number of training samples from the 50 poisoned samples instead of all samples.

Results. In the exploitation experiment, we first identified the state q_i of a given input, which is most influenced by the 50 training samples injected with "NeurIPS", *i.e.*, $\mathcal{I}(q_i, \text{"NeurIPS"})$ that has the largest size. Then, we insert the word in the position right after q_i. The baseline randomly selects one position to insert "NeurIPS". In the detection experiment, conversely, we identified the state q_i before "NeurIPS" from the failed input. With the transition influence function $\mathcal{I}(q_i, \text{"NeurIPS"})$, we could find a set of training samples and then retrain the model by removing the identified samples. We also selected the same number of samples from the 50 modified training samples rather than all the training data for the random baseline. We randomly selected 100 positive test comments to conduct the two experiments, and each experiment is repeated for 10 times. More details can be found in supplementary.

Table 6 shows the average results of the backdoor attack and detection. Our method can identify the most influenced position and achieve higher attack accuracy (70.2%) than random insertion (49.9%). For failed inputs, our method identities 4.4 (Column *#Re*) most influential training samples on average. After removing them and retraining the model, 72.9% failed inputs could be correctly handled, while only 37.6% failed inputs are repaired using the random strategy. The results demonstrate that not all modified samples are responsible for a failed input, and the segment-level analysis could identify the influential training samples precisely.

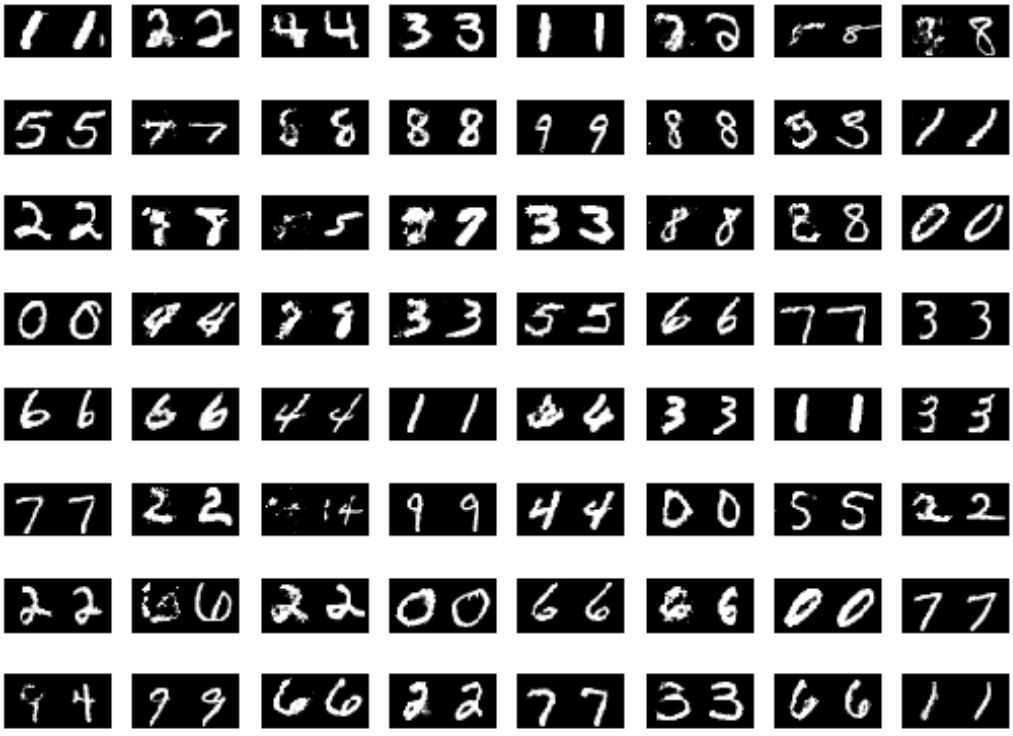


Figure 8: Reverse Example 1: left images are reversed based on the feature of the right images.

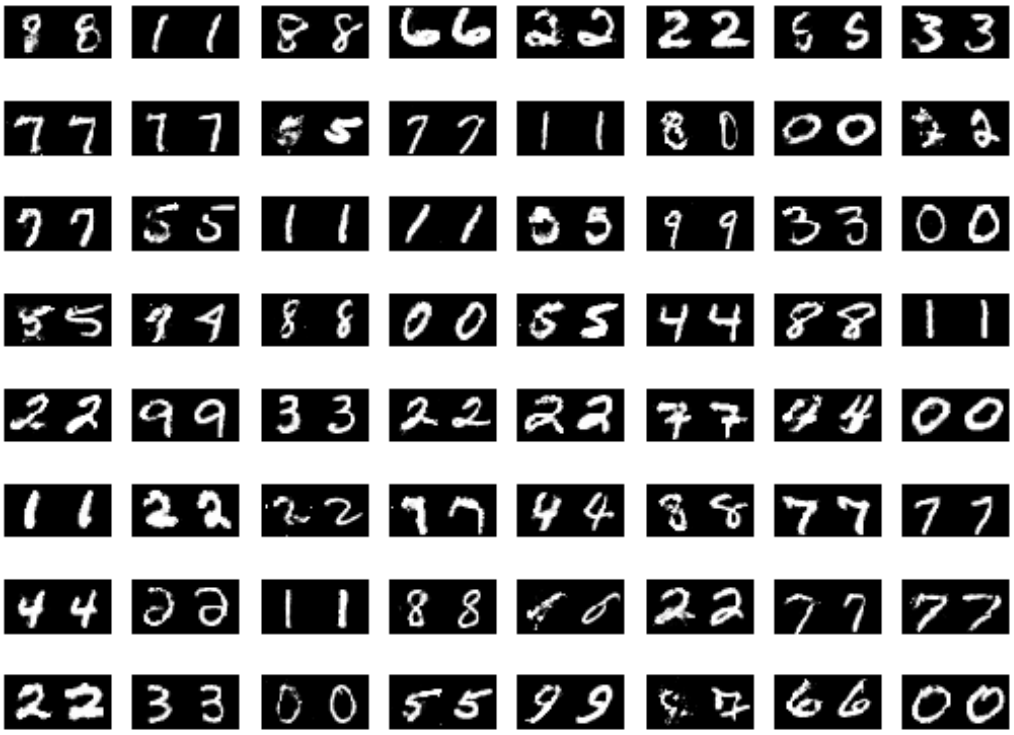


Figure 9: Reverse Example 2: left images are reversed based on the feature of the right images.

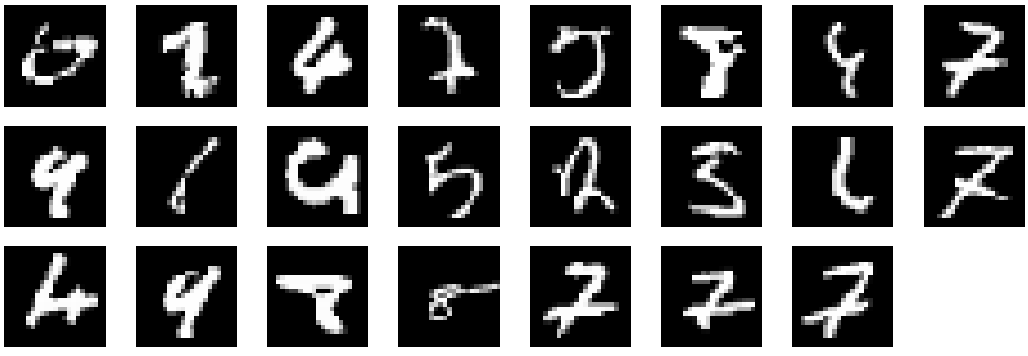


Figure 10: Failed images

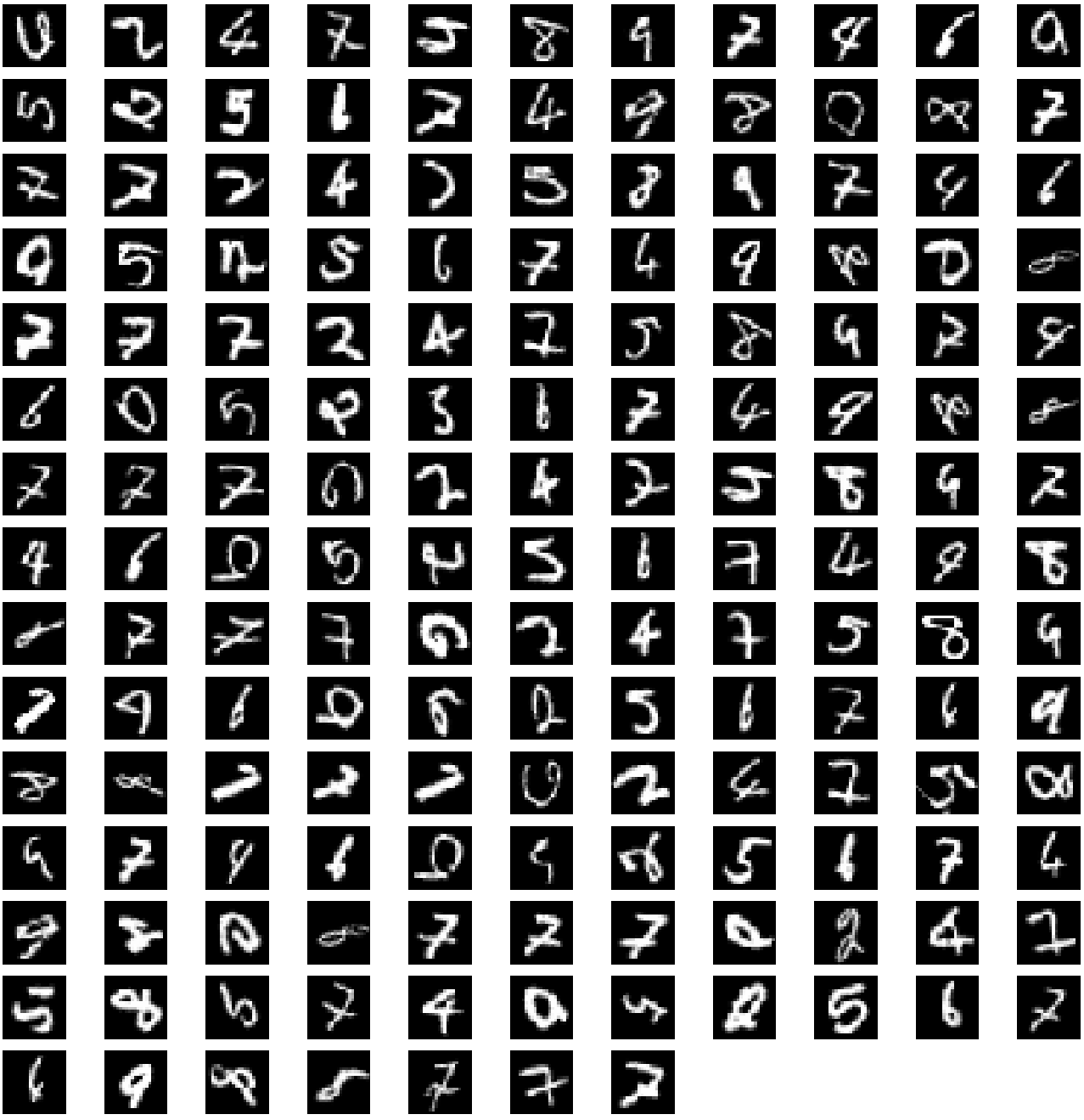


Figure 11: Generated images for repairing the faults