Supplementary Material

A. Experimental Settings and Examples

A.1. Implementation Details

Due to the state vector \( (i.e., h) \) has a high-dimensionality, we reduced it to a lower dimensionality \( (i.e, \text{ten}) \) through the Principal Component Analysis (PCA). For the image classification, we adopt the simplest input abstraction \( \alpha : x_i \rightarrow \epsilon \), which abstracts every segment as \( \epsilon \), \( i.e., \sum = \{ \epsilon \} \). 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, \( i.e., \text{"toxic"}, \text{"severe toxic"}, \text{"obscene"}, \text{"threat"}, \text{"insult"}, \text{"identity hate"} \) and the normal ones \( i.e., \text{non-toxic} \). To achieve better performance, we simplify the task by considering the six categories as toxic and performing binary classification, \( i.e., \text{whether a given comment is toxic or not} \). 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 \( 28 \times 28 \), the LSTM size is \( 28 \times 100 \) \( i.e., \) at each time, LSTM reads one row of the image \) and the size of the output layer is \( 100 \times 10 \). 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 \( 300 \times 300 \) \( i.e., \) at each time, the GRU model processes each word \) and the size of the output layer is \( 300 \times 2 \) \( i.e., \) 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 \( 28 \times 12 \), for each row, we have a 12-dimensional feature vector \( i.e., f_i \text{ in Definition 5} \) including 10-dimensional confidence score, 1 unique identifier and 1 prediction label. The output layer is a liner layer whose size is \( 28 \times 12 \times 10 \). For Toxic, the input size is \( 200 \times 12 \). 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 \( 200 \times 12 \times 2 \). 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 \( \text{(Hara et al., 2019)} \), we randomly selected 30\% of the images in MNIST dataset and flipped their labels from 7 to 1. Since the comparison baseline SGD \( \text{(Hara et al., 2019)} \) 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 \( \text{https://github.com/sato9hara/sgd-influence} \) 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\%, \ldots, 100\% training samples, and then retrained...
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 these additional experiments to validate the effectiveness of our refinement process, the correctness of our model-based analysis, 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 Stabe 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 Stabe 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...
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 $x$, we use $t_x$ and $m_x$ 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 $\phi^n_n$) that are most responsible for the misclassification of $x$. We use $T_{t_x}$ and $T_{m_x}$ to denote the training samples in $\phi^n_n$, whose ground truth labels are $t_x$ and $m_x$, respectively.

With the identified responsible data $\phi^n_n$, we define the following metrics to understand the behaviors of the failed prediction.

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

Our evaluation results demonstrate that the feature of $x$ is more similar to the features of the data whose truth labels are $m_x$ in $\phi^n_n$. Hence, the model is classified as $m_x$ incorrectly.

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

![Figure 4: The refinement results with different PCA configurations](image-url)
Table 5: Feature Accuracy on IMDB.

<table>
<thead>
<tr>
<th></th>
<th>R_L</th>
<th>ID</th>
<th>(ID, R_L)</th>
<th>CSs</th>
<th>(ID, R, L, CSs)</th>
<th>Orig</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>67.86</td>
<td>79.98</td>
<td>86.56</td>
<td>86.09</td>
<td><strong>86.74</strong></td>
<td>89.80</td>
</tr>
</tbody>
</table>

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 Figure 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.
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
We randomly selected 100 test samples that are predicted we retrained a new model by removing the identified sam-
ple. We identified the influential training samples from which contains one word “NeurIPS”, we got the trace
we have (see Equation 2), we select the number of the clustering quantities 4.4 (Column #Re) most influential training samples
as negative after the insertion. 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.

<table>
<thead>
<tr>
<th>Attack Rate</th>
<th>Fix Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Rand</td>
<td>Our Rand</td>
</tr>
<tr>
<td>70.2% 49.9%</td>
<td>72.9% 37.6%</td>
</tr>
</tbody>
</table>

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 influ-
tential training samples I(q_i, “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 misclassi-
cified. For one test sample \( x = (x_1, \ldots, x_n) \), we got its trace \( \tau_x = (q_0, x_1, q_1, \ldots, x_n, q_n) \) 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 \( I(q_i, “NeurIPS”) \). We inserted the target word to the position \( i = \arg \max_{0 \leq i \leq n} |I(q_i, “NeurIPS”)| \). As the base-
line, 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 influ-
tential for this test sample. Given the misclassi-
cified test sample \( x = (x_1, \ldots, “NeurIPS”, \ldots, x_n) \) which contains one word “NeurIPS”, we got the trace \( \tau_x = (q_0, x_1, \ldots, q_i, “NeurIPS”, q_{i+1}, x_n, q_n) \). Then, we identified the influential training samples from \( I(q_i, “NeurIPS”) \). Note that we may find multiple influ-
tential 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 sam-
ple and checked whether the misclassified input could be repaired. For random baseline, to be fair, we randomly se-
lected the same number of training samples from the 50 poisoned samples instead of all samples.

**Results.** In the exploitation experiment, we first identi-
fied the state \( q_i \) of a given input, which is most influenced by the 50 training samples injected with “NeurIPS”, i.e.,
\( I(q_i, “NeurIPS”) \) 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 \( I(q_i, “NeurIPS”) \), 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 ex-
periments, 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 ran-
dom insertion (49.9%). For failed inputs, our method iden-
tifies 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