Automatic Testing and Improvement of Machine Translation

Zeyu Sun
Key Laboratory of HCST
Peking University, MoE
szy._@pku.edu.cn

Jie M. Zhang∗
University College London
jie.zhang@ucl.ac.uk

Mark Harman
Facebook London
University College London
mark.harman@ucl.ac.uk

Mike Papadakis
University of Luxembourg
mike.papadakis@gmail.com

Lu Zhang
Key Laboratory of HCST
Peking University, MoE
zhanglucs@pku.edu.cn

ABSTRACT
This paper presents TransRepair, a fully automatic approach for testing and repairing the consistency of machine translation systems. TransRepair combines mutation with metamorphic testing to detect inconsistency bugs (without access to human oracles). It then adopts probability-reference or cross-reference to post-process the translations, in a grey-box or black-box manner, to repair the inconsistencies. Our evaluation on two state-of-the-art translators, Google Translate and Transformer, indicates that TransRepair has a high precision (99%) on generating input pairs with consistent translations. With these tests, using automatic consistency metrics and manual assessment, we find that Google Translate and Transformer have approximately 36% and 40% inconsistency bugs. Black-box repair fixes 28% and 19% bugs on average for Google Translate and Transformer. Grey-box repair fixes 30% bugs on average for Transformer. Manual inspection indicates that the translations repaired by our approach improve consistency in 87% of cases (degrading it in 2%), and that our repairs have better translation acceptability in 27% of the cases (worse in 8%).

KEYWORDS
machine translation, testing and repair, translation consistency

1 INTRODUCTION
Machine learning has been successful in providing general-purpose natural language translation systems, with many systems able to translate between thousands of pairs of languages effectively in real time [19]. Nevertheless, such translation systems are not perfect and the bugs that users experience have a different character from those on traditional, non-machine learning-based, software systems [2, 26, 27, 56].

The consequences of mistranslation have long been studied and their effects have been shown to be serious. For example, the infamous historic mistranslation of Article 17 of the Treaty of Uccialli reportedly led to war [11]. Such truly profound and far-reaching consequences of mistranslation are also reportedly becoming a serious and potent source of international tension and conflict [31].

The consequences of mistranslation through machine-based translators have also been shown to be serious. For example, machine translations have been shown to exhibit pernicious fairness bugs that disproportionately harm specific user constituencies [34]. We have found such examples of fairness bugs in widely used industrial strength translation systems. Figure 1 shows several such Google Translate results for the language pair (English → Chinese)1. As can be seen from the figure, Google Translate translates ‘good’ into ‘hen hao de’ (which means ‘very good’) when the subject is ‘men’ or ‘male students’. However, interestingly, but also sadly, it translates ‘good’ into ‘hen duo’ (which means ‘a lot’) when the subject is ‘women’ or ‘female students’2.

Such inconsistency may confuse users, and is also clearly unfair to female researchers in computer science; producing ‘a lot’ of research is clearly a more pejorative interpretation, when compared to producing ‘very good’ research. To avoid such unfair translations (at scale), we need techniques that can automatically identify and remedy such inconsistencies.

<table>
<thead>
<tr>
<th>English</th>
<th>Chinese (Google Translation)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Men do good research in computer science.</td>
<td>Nānren zài jìsuàn jì fēngxiàn zuòdè hén hǎo de yànyǔ</td>
<td>good → hén hào de (very good)</td>
</tr>
<tr>
<td>Women do good research in computer science.</td>
<td>Nüxing zài jìsuàn jì fēngxiàn zuòdè hén duo yànyǔ</td>
<td>good → hén duo (a lot)</td>
</tr>
<tr>
<td>Male students do good research in computer science.</td>
<td>Nán xuéshēng zài jìsuàn jì fēngxiàn zuòdè hén hǎo de yànyǔ</td>
<td>good → hén hào de (very good)</td>
</tr>
<tr>
<td>Female students do good research in computer science.</td>
<td>Nǚ xuéshēng zài jìsuàn jì fēngxiàn zuòdè hén duo yànyǔ</td>
<td>good → hén duo (a lot)</td>
</tr>
</tbody>
</table>

Figure 1: Examples of fairness issues brought by translation inconsistency (from Google Translate)

To tackle this problem, we introduce a combined testing and repair approach that automatically generates tests for real-world machine translation systems, and automatically repairs the mistranslations found in the testing phase. As we show in this paper, we need to rethink the conventional approaches to both testing and repair in order to apply them to natural language translation.

Existing work has tested whether machine translation systems provide stable translations for semantically-equivalent transformations, 1The four translations were obtained on 23rd July, 2019. These examples are purely for illustration purposes, and are not intended as a criticism of Google Translate. It is likely that other mainstream translation technologies will have similar issues. 2Similar issues also exist in translations between other languages. With a cursory check, we already found a case with German—→Chinese.
such as synonym replacement (e.g., buy→purchase) [5] or abbreviation replacement (e.g., what’s→what is) [36]. However, no previous work has focused on the testing and repair of translation inconsistency regarding context-similar transformation; the transformation between sentences that have similar word embeddings [12] yet share context in the corpus (e.g., simple gender-based transformations, such as boys→girls).

In order to tackle the testing problem, we introduce an approach that combines mutation [22, 32, 54, 55] with metamorphic testing [3, 52]. The approach conducts context-similar mutation to generate mutated sentences that can be used as test inputs for the translator under test. When a context-similar mutation yields above-threshold disruption to the translation of the non-mutated part, the approach reports an inconsistency bug.

Traditional approaches to ‘repairing’ machine learning systems typically use data augmentation or algorithm optimisation. These approaches can best be characterised as to “improve” the overall effectiveness of the machine learner, rather than specific repairs for individual bugs; they also need data collection/labelling and model retraining, which usually have a high cost.

Traditional approaches to ‘repairing’ software bugs are white box, because the techniques need to identify the line(s) of source code that need(s) to be modified in order to implement a fix. Such approaches inherently cannot be applied to fix software for which source code is unavailable, such as third-party code.

Our insight is that by combining the results of repeated (and potentially inconsistent) output from a system, we can implement a light-weight black-box repair technique as a kind of ‘post-processing’ phase that targets specific bugs. Our approach is the first repair technique to repair a system in a purely black-box manner. We believe that black-box repair has considerable potential benefits, beyond the specific application of machine translation repair. It is the only available approach when the software engineer is presented with bugs in systems for which no source code is available.

We demonstrate not only that black-box repair is feasible, but that it can scale to real world industrial-strength translation systems, such as Google Translate. We also present results for grey-box repair for which the predictive probability is available.

TransRepair is evaluated on two state-of-the-art machine translation systems, Google Translate and Transformer [46]. In particular, we focus on the translation between the top-two most widely-spoken languages: English and Chinese. These languages each have over one billion speakers worldwide [8]. Nevertheless, only 10 million people in China (less than 1% of the population) are able to communicate via English [47, 48]. Since so few people are able to speak both languages, machine translation is often attractive and sometimes necessary and unavoidable.

Our results indicate that TransRepair generates valid test inputs effectively with a precision of 99%; 2) TransRepair automatically reports inconsistency bugs effectively with the learnt thresholds, with a mean F-measure of 0.82/0.88 for Google Translate/Transformer; 3) Both Google Translate and Transformer have inconsistency bugs. Automated consistency metrics and manual inspection reveal that Google Translate has approximately 36% inconsistent translations on our generated test inputs. 4) Black-box repair reduces 28% and 19% of the bugs of Google Translate and Transformer. Grey-box reduces 30% of the Transformer bugs. Manual inspection indicates that the repaired translations improve consistency in 87% of the cases (reducing it in only 2%), and have better translation acceptability in 27% of the cases (worse in only 8%).

## 2 APPROACH

This section introduces the overview and the details of each step for TransRepair.

### 2.1 Overview

A high level view of TransRepair is presented in Figure 2. From this Figure it can be seen that TransRepair automatically tests and repairs the inconsistency of machine translation based on the following three major steps:

1) **Automatic test input generation.** This step generates transformed sentences (test inputs) to be used for consistency testing. For each sentence, TransRepair conducts sentence mutations via context-similar word replacement. The generated mutant candidates are filtered using a grammar check. The mutants that pass the grammar check are then regarded as the final test inputs for the machine translator under test. Details are presented in Section 2.2.

2) **Automatic test oracle generation.** This step introduces the generation of oracles, which are used for identifying inconsistent translations (bugs). In this step, we rely on the metomorphic relationship between translation inputs and translation outputs. The idea is that translation outputs from both the original sentence and its context-similar mutant(s) should have a certain degree of consistency modulo the mutated word. We use similarity metrics that measure the degree of consistency between translated outputs as test oracles. Details of this step are presented in Section 2.3. We explore four similarity metrics, which are described in Section 3.2.

3) **Automatic inconsistency repair.** This step automatically repairs the inconsistent translation. TransRepair applies black-box and grey-box approaches, which transform the original translation based on the best translation among the mutants. We explore two ways of choosing the best translation, one using predictive probability, the other using cross-reference. Details of this step are given in Section 2.4.

### 2.2 Automatic Test Input Generation

The input generation process contains the following steps.

#### 2.2.1 Context-similarity Corpus Building

To conduct context-similar word replacement, the key step is to find a word(s) that can be replaced with other(s) (similar ones) without hurting the sentence structure. The new sentence generated via word replacement should yield consistent translations with the original.

Word vectors capture the meaning of a word through their context [35]. To measure the similarity, we use word vectors trained from text corpora. In our approach, the word similarity between two words $w_1$ and $w_2$, denoted by $\text{sim}(w_1, w_2)$, is computed by the formula below, where $v_x$ denotes the vector of the word $x$.

$$\text{sim}(w_1, w_2) = \frac{\sigma_{w_1} \cdot \sigma_{w_2}}{|\sigma_{w_1}| \cdot |\sigma_{w_2}|} \tag{1}$$

### 2.2.2 Automatic test oracle generation

This step automatically generates oracles, which are used for identifying inconsistent translations (bugs). In this step, we rely on the metomorphic relationship between translation inputs and translation outputs. The idea is that translation outputs from both the original sentence and its context-similar mutant(s) should have a certain degree of consistency modulo the mutated word. We use similarity metrics that measure the degree of consistency between translated outputs as test oracles. Details of this step are presented in Section 2.3. We explore four similarity metrics, which are described in Section 3.2.

### 2.2.3 Automatic inconsistency repair

This step automatically repairs the inconsistent translation. TransRepair applies black-box and grey-box approaches, which transform the original translation based on the best translation among the mutants. We explore two ways of choosing the best translation, one using predictive probability, the other using cross-reference. Details of this step are given in Section 2.4.

### 2.3 Similarity Metrics

This section introduces the overview and the details of each step for TransRepair.

#### 2.3.1 METRIC 1: JACCARD SIMILARITY

The Jaccard similarity is a similarity metric that measures the similarity between two sets. It is the size of the intersection divided by the size of the union of the sets.

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

#### 2.3.2 METRIC 2: TANIMOTO SIMILARITY

The Tanimoto similarity is a similarity metric that measures the similarity between two sets. It is the size of the intersection divided by the size of the union of the sets.

$$\text{Tanimoto}(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

#### 2.3.3 METRIC 3: THRESHOLDED JACCARD SIMILARITY

This metric is similar to the Jaccard similarity, but it is thresholded to only consider translations that are above a certain threshold.

#### 2.3.4 METRIC 4: THRESHOLDED TANIMOTO SIMILARITY

This metric is similar to the Tanimoto similarity, but it is thresholded to only consider translations that are above a certain threshold.

### 2.4 Repair Approach

This section introduces the overview and the details of each step for TransRepair.

#### 2.4.1 Black-box Repair

Black-box repair is feasible, but it can scale to real world industrial-strength translation systems, such as Google Translate. We also present results for grey-box repair for which the predictive probability is available. TransRepair is evaluated on two state-of-the-art machine translation systems, Google Translate and Transformer [46]. In particular, we focus on the translation between the top-two most widely-spoken languages: English and Chinese. These languages each have over one billion speakers worldwide [8]. Nevertheless, only 10 million people in China (less than 1% of the population) are able to communicate via English [47, 48]. Since so few people are able to speak both languages, machine translation is often attractive and sometimes necessary and unavoidable.

Our results indicate that TransRepair generates valid test inputs effectively with a precision of 99%; 2) TransRepair automatically reports inconsistency bugs effectively with the learnt thresholds, with a mean F-measure of 0.82/0.88 for Google Translate/Transformer; 3) Both Google Translate and Transformer have inconsistency bugs. Automated consistency metrics and manual inspection reveal that Google Translate has approximately 36% inconsistent translations on our generated test inputs. 4) Black-box repair reduces 28% and 19% of the bugs of Google Translate and Transformer. Grey-box reduces 30% of the Transformer bugs. Manual inspection indicates that the repaired translations improve consistency in 87% of the cases (reducing it in only 2%), and have better translation acceptability in 27% of the cases (worse in only 8%).
To construct a reliable context-similar corpus, we take two word-vector models and use the intersection of their trained results. The first model is GloVe [35], which is trained from Wikipedia 2014 data [49] and GigaWord 5 [38]. The second model is SpaCy [40], which is a multi-task CNN trained on OntoNotes [1] including the data collected from telephone conversations, newswire, newsgroups, broadcast news, broadcast conversation, and weblogs. When two words have a similarity of over 0.9 for both models, we deem the word pair to be context-similar and place it in the context-similarity corpus. In total, we collected 131,933 word pairs using this approach.

### 2.2.2 Translation Input Mutation.

We introduce word replacement and structural filtering respectively in the following.

**Word replacement.** For each word in the original sentence, we search to determine whether there is a match in our corpus. If we find a match, we replace the word with its context-similar one and generate the resulting mutated input sentence. Compared with the original sentence, each mutated sentence contains a single replaced word. To reduce the possibility of generating mutants that do not parse, we only replace nouns, adjectives, and numbers.

**Structural filtering.** The generated mutated sentence may fail to parse, because the replaced word may not fit the context of the new sentence. For example, “one” and “another” are context-similar words, but “a good one” parses, while “a good another” does not. To address such parsing failures, we apply additional constraints to sanity check the generated mutants. In particular, we apply structural filtering, based on the Stanford Parser [30]. Suppose the original sentence is $s = w_1, w_2, ..., w_i, ..., w_n$, the mutated sentence is $s' = w_1, w_2, ..., w_i', ..., w_n$, where $w_i$ in $s$ is replaced with $w_i'$ in $s'$. For each sentence, the Stanford Parser outputs $l(w_i)$, the part-of-speech tag of each word used in the Penn Treebank Project [44]. If $l(w_i) \neq l(w_i')$, we remove $s'$ from the mutant candidates because the mutation yields changes in the syntactic structure.

We manually inspect the quality of the generated inputs and report results in Section 4.

### 2.3 Automatic Test Oracle Generation

To perform testing we need to augment the test inputs we generate with test oracles, i.e., predicates that check whether an inconsistency bug has been found. To do so, we assume that the unchanged parts of the sentences preserve their adequacy and fluency modulo the mutated word. Adequacy means whether the translation conveys identical meaning, whether there is information lost, added, or distorted; Fluency means whether the output is fluent and grammatically correct [7, 15].

Let $t(s)$ and $t(s')$ be the translations of sentences $s$ and $s'$ that were produced by replacing the word $w$ (in $s$) with $w'$ (in $s'$). Thus, we would like to check the similarity between $t(s)$ and $t(s')$ when ignoring the translations for words $w$ and $w'$. Unfortunately, it is not easy to strip the effect of $w$ and $w'$, because $a) w$ and $w'$ may change the entire translation of the sentences, and $b)$ it is not easy to accurately map the words $w$ and $w'$ with their respective one(s) in the translated text.

To bypass this problem, we calculate the similarity of subsequences of $t(s)$ and $t(s')$, and use the largest similarity to approximate the consistency level between $t(s)$ and $t(s')$. Algorithm 1 shows the process. For $t(s)$ and $t(s')$, we first apply GNU Wdiff [10] to get the difference slices (Line 1). GNU Wdiff compares sentences on word basis, and is useful for comparing two texts in which a few words have been changed [10]. With Wdiff, the difference slices of two sentences “A B C D F” and “B B C G H F” are represented as “A”, “D” and “B”, “G H” for the two sentences, respectively.

The diff slices of $t(s)$ and $t(s')$ are saved to set $B_s$ and $B_{s'}$. We then delete a slice from the translations, one at a time (Line 5 and Line 9). Each slice corresponds to one subsequence with this slice deleted. For the example above, “A B C D F” will have two subsequences: “B C D F” (deleting “A”) and “A B C F” (deleting “D”). These new subsequences of $t(s)$/$t(s')$ are added into set $T_0/T_m$ (Line 6 and Line 10).

For each element in set $T_0$, we compute its similarity$^5$ with each element in the set $T_m$ (Line 15). Thus, we get $|T_0| \times |T_m|$ similarity scores, where $|T_0|$ and $|T_m|$ is the size of $T_0$ and $T_m$. We then use the highest similarity as the result of the final consistency score (Lines 16).

This configuration reduces the influence of the mutated word, and helps to select an inconsistency upper bound. Even if the two subsequences with the largest similarity contain the replaced word, other sentence parts have worse similarity, so it is unlikely that this case is biased by the replaced word (leads to false positives).

Details about the experimental setup and the results of the threshold setup are discussed in Section 4.2. Additionally, we evaluate the

---

$^5$Long slices are unlikely to correspond to the mutated word, we thus only keep slices that are no longer than 5 words.

$^6$We explore four types of similarity metrics in this paper (see full details in Section 3.2).
effectiveness of our consistency measurements via manual inspection. These results are presented in Section 4.2.

Algorithm 1: Process of obtaining consistency score

Data: $t(s)$: translation of the original sentence; $t(s')$: translation of the mutant

Result: ConScore: Consistency score between $t(s)$ and $t(s')$

1. $B_s, B_{s'} = \text{Wdiff}(t(s), t(s'))$
2. $T_0 = \{ t(s) \}$
3. $T_m = \{ t(s') \}$
4. For each subsequence $b_s \in B_s$, do
   5. $r = \text{DeleteSub}(t(s), b_s)$
   6. $T_o = T_o \cup \{ r \}$
7. End
8. For each subsequence $b_{s'} \in B_{s'}$, do
   9. $r' = \text{DeleteSub}(t(s'), b_{s'})$
   10. $T_m = T_m \cup \{ r' \}$
End
11. ConScore = -1
12. For each sentence $a \in T_o$, do
13.   For each sentence $b \in T_m$, do
14.     Sim = $\text{ComputeSimilarity}(a, b)$
15.     ConScore = Max(ConScore, Sim)
16. End
17. End
18. Return ConScore

2.4 Automatic Inconsistency Repair

We first introduce the overall repair process, then introduce two mutant translation ranking approaches (probability and cross-reference).

2.4.1 Overall Repair Process. First, we repair the translation of the original sentences and then we seek to find a translation for the mutant, which must pass our consistency test.

Algorithm 2 shows the repair process. For $t(s)$ which has been revealed to have inconsistency bug(s), we generate a set of mutants and get their translations $t(s_1), t(s_2), ..., t(s_n)$. These mutants and their translations, together with the original sentence and its translation, are put into a dictionary, $T$ (Line 1). We then rank the elements in $T$, in descending order, using the predictive probability or cross-reference, and put the results in OrderedList (Line 2). The details of probability and cross-reference ranking are given in Section 2.4.2 and Section 2.4.3.

Next, we apply word alignment to obtain the mapped words between $s$ and $t(s)$ as $a(s)$ (Line 3). Word alignment is a natural language processing technique that connects two words if and only if they have a translation relationship. In particular, we adopt the technique proposed by Liu et al. [29], which uses a latent-variable log-linear model for unsupervised word alignment. We then check whether a sentence pair $(s_r, t(s_r))$ in OrderedList can be adopted to repair the original translation. We follow the ranking order, until we find one mutant translation that is acceptable for inconsistency repair.

If $s_r$ is the original sentence ($s_r = s$), it means the original translation is deemed a better choice than other mutant translations and so we will not touch it (Lines 6-8). Otherwise, we do the same alignment to $s_1$ and $t(s_1)$ as to $s$ and $t(s)$. The variables $w_i, w'_i$ denote the replaced words in $s$, $s_r$ and we get the translated words $t(w_i), t(w'_i)$ through the alignment (Lines 9-12).

Word alignment is not 100% accurate. If we directly map the translation by replacing $t(w'_i)$ with $t(w_i)$, it may lead to grammatical errors or context mismatches. We adopt the following strategies to judge whether the replacement is acceptable. 1) We constrain that $w_i, w'_i$ and $t(w_i), t(w'_i)$ must belong to the same type (i.e., numeric or non-numeric) (Line 13-15). 2) If the replaced words are of the non-numeric type, we apply Stanford Parser to check whether the replacement would lead to structural changes (Line 17-21).

When repairing the translation of the mutated input (Line 22), we get the repaired result of the original sentence (Line 23), then check whether the translation solution candidate is consistent with the repaired translation of the original sentence (Line 24-26). If not, we proceed by checking other repair candidates.

2.4.2 Translation Ranking based on Probability. For a sentence $s$ and its mutants $S = s_1, s_2, ..., s_n$, let $t(s)$ be the translation of $s$, let
t(s_i) be the translation of mutant s_i. This approach records the translation probability for each t(s_i), and chooses the mutant with the highest probability as a translation mapping candidate. The translation of the corresponding mutant will then be mapped back to generate the final repaired translation for s using word alignment.

This is a grey-box repair approach. It requires neither the training data nor the source code of the training algorithm, but needs the predictive probability provided by the machine translator. We call this grey-box because implementors may regard this probability information as an internal attribute of the approach, not normally intended to be available to end users.

2.4.3 Translation Ranking based on Cross-reference. For a sentence s and its mutants S = s_1, s_2, ..., s_n, let t(s) be the translation of s, and let t(s_i) be the translation of mutant s_i. This approach calculates the similarity among t(s), t(s_1), t(s_2), ..., t(s_n), and uses the translation that maps the best (with the largest mean similarity score) with other translations to map back and repair the previous translation.

This is a black-box repair approach. It requires only the ability to execute the translator under test and the translation outputs.

3 EXPERIMENTAL SETUP

In this section, we introduce the experimental setup that evaluates the test input generation, translation inconsistency detection, and translation inconsistency repair.

3.1 Research questions

We start our study by assessing the ability of TransRepair to generate valid and consistent test inputs that can be adopted for consistency testing. Hence we ask:

RQ1: How accurate are the test inputs of TransRepair?

We answer this question by randomly sampling some candidate pairs and checking (manually) whether they are valid. The answer to this question ensures that, TransRepair, indeed generates inputs that are suitable for consistency checking.

Given that we found evidence that TransRepair generates effective test pairs, we turn our attention to the question of how effective these pairs are at detecting consistency bugs. Therefore we ask:

RQ2: What is the bug-revealing ability of TransRepair?

To answer RQ2 we calculate consistency scores based on similarity metrics to act as test oracles (that determine whether a bug has been detected). To assess the bug-revealing ability of the TransRepair, we manually check a sample of translations and compare the resulting manual inspection results with automated test results.

Having experimented with fault revelation, we evaluate the repair ability of TransRepair to see how well it repairs inconsistency bugs. Thus, we ask:

RQ3: What is the bug-repair ability of TransRepair?

To answer this question, we record how many inconsistency bugs are repaired (assessed by consistency metrics and manual inspection). We also manually examine the translations repaired by TransRepair, and check whether they improve translation consistency as well as quality.

3.2 Consistency Metrics

We explore four widely-adopted similarity metrics for measuring inconsistency. For ease of illustration, we use t_1 to denote the translation output of the original translation input; we use t_2 to denote the translation output of the mutated translation input.

1. LCS-based metric. It measures the similarity via normalised length of a longest common subsequence between t_1 and t_2:

   \[ M_{LCS} = \frac{\text{len}(\text{LCS}(t_1, t_2))}{\text{Max}(\text{len}(t_1), \text{len}(t_2))} \]

2. ED-based metric. This metric is based on the edit distance between t_1 and t_2. Edit distance is a way of quantifying how dissimilar two strings are by counting the minimum number of operations required to transform one string into the other [37]. To normalise the edit distance, we use the following formula which has also been adopted in previous work [16, 53].

   \[ M_{ED} = 1 - \frac{\text{ED}(t_1, t_2)}{\text{Max}(\text{len}(t_1), \text{len}(t_2))} \]

3. tf-idf-based metric. tf-idf (term frequency–inverse document frequency) can be used to measure similarity in terms of word frequency. Each word w has a weighting factor, which is calculated based on the following formula, where C is a text corpus (in this paper we use the training data of Transformer), |C| is the sentence number in C, f_w is the number of sentences that contain w.

   \[ w_{idf} = \log((|C| + 1)/(f_w + 1)) \]

4. BLEU-based metric. The BLEU (BiLingual Evaluation Understudy) is an algorithm that automatically evaluates machine translation quality via checking the correspondence between a machine’s output and that of a human. It can also be adopted to compute the similarity between the translation of the original sentence and the translation of a mutant. Details, description, and motivation for the BLEU score can be found in the translation literature [33]. Due to lack of space we only provide an overview here.

BLEU first counts the number of matched sub-strings between sentences and computes a precision p_n (which is called modified n-gram precision [33], where n means the sub-sequences length). For example, in sentences “A A B C” (s_1) and “A B B C” (s_2) there are three 2-gram subsequences in s_2: AB, BB, and BC. Two of them are matched with those from s_1: AB and BC. Thus, p_2 is 2/3.
As well as $p_n$, the calculation of BLEU score also requires an exponential brevity penalty factor $BP$ (to penalise overly short translations), which is shown by Formula 5. $c$ denotes the length of $t(s_j)$ and $r$ is the length of $t(s)$.

$$BP = \begin{cases} 
1 & \text{if } c > r \\
\frac{1}{e^{(1-r/c)}} & \text{if } c \leq r
\end{cases}$$

(5)

The BLEU score is finally computed by Formula 6, where $w_n = \frac{1}{N}$ (we use $N = 4$ in this paper) is the uniform weights.

$$BLEU = BP \cdot \exp \left( \sum_{n=1}^{N} w_n \log p_n \right).$$

(6)

Since BLEU is unidirectional (i.e., $BLEU(s, s') \neq BLEU(s', s)$), we use the higher score for measuring the similarity between $s$ and $s'$. This is consistent with our intention in Algorithm 1: to get an upper bound of the consistency, thereby avoiding false positive claims about translation bugs.

### 3.3 Machine Translators

Our experiment considers both industrial and state-of-the-art research-oriented machine translators. One is Google Translate [14] (abbreviated as GT in the results section), a widely used machine translation service developed by Google. The other is Transformer [46], a translator studied by the research community.

We use Google translate, because it is an example of a system that forces us to perform black-box repairs; we have no access to the training data nor the code of the translation system, and therefore any improvements, by definition, can only have been achieved by a black-box approach. Also, of course, it is a production-quality mainstream translation system, making the results more interesting.

We use the default setup to train Transformer. Transformer is trained based on three datasets: the CWMT dataset [6] with 7,086,820 parallel sentences, the UN dataset [59] with 15,886,041 parallel sentences, and the News Commentary dataset [50] with 252,777 parallel sentences as the training data. The validation data (to help tune hyper-parameters) is also from the News Commentary and contains 2,002 parallel sentences. Transformer runs with the TensorFlow deep learning library [45]. To get a more effective translator, we train the model for 500,000 epochs.

### 3.4 Test Set

Following previous machine translation research [17, 18], we use a test set from the News Commentary [50] dataset for both Google Translate and Transformer. The test set contains 2,001 parallel sentences and are different from the training set and validation set. The Chinese sentences in our experiments are in the form of characters.

Our experiments were conducted on Ubuntu 16.04 with 256GB RAM and four Intel E5-2620 v4 CPUs (2.10 GHz), which contains 32 cores all together. The neural networks we used were all trained on a single Nvidia Titan RTX (24 GB memory).

## 4 RESULTS

This section reports the results that answer our research questions.

---

### 4.1 Effectiveness on Input Generation (RQ1)

We start by answering RQ1. For each test sentence, we generate mutants and check whether they pass the structural filtering (see more in Section 2.2.2). In particular, for each sentence we generate up to 5 mutants that pass through the filter (we study the influence of the number of mutants in Section 5). For the 2,001 test sentences, 21,960 mutant candidates are generated, with 17,268 discarded by structural filtering. In the rest of our experiment we use the remaining 4,692 mutants, which are paired with 1,323 sentences, as test inputs.

To manually assess whether these test inputs are qualified for detecting translation inconsistency, we randomly sampled 400 of them. The first two authors manually checked the validity of the inputs, i.e., whether the replaced word in the mutant leads to grammatical errors and whether the mutant ought to have consistent translations with the original sentence. This validation step reveals three invalid mutants: 1) He was a kind spirit with a big heart: kind → sort; 2) Two earthquakes with magnitude 4.4 and 4.5 respectively: Two → Six; 3) It is in itself a great shame: great → good.

The remaining 397 of the 400 meet the two validity criteria, indicating a precision of 99%. We conclude that our two strategies for the intersection of two word2vec models, and the use of Stanford Parser as a filter have a high probability of yielding valid test sentences. The 400 mutants and the manual assessment results can be found on the TransRepair homepage [43].

In the next section we use the 4,692 mutants (from the 1,323 original sentences) to examine the test translation consistency of machine translation systems.

---

### 4.2 Inconsistency-revealing Ability of TransRepair (RQ2)

This section answers RQ2, i.e., investigates the inconsistency-revealing ability of TransRepair. To answer this question, we investigate: 1) the consistency metric values between the mutant and the original translations; 2) the manual inspection results of translation inconsistency. We also explore how close the consistency metrics and manual inspection are in evaluating inconsistency.

**Consistency Metric Values.** We translate the 4,692 generated inputs with Google Translate and Transformer, and compare them with the translations of the original sentences, following the steps of Algorithm 1. For each mutant, we calculate four consistency scores, each one corresponding to one of the similarity metrics (outlined in Section 3.2).

Figure 3 shows the histogram of consistency scores that are lower than 1.0. As can be seen from the figure, different metric values have different distributions, yet overall, all the four metrics report a large number of translations (i.e., around 47% of the total translations) with a score below 1.0, indicating the presence of translation inconsistency.

---

Answer to RQ1: TransRepair has good precision (99%) for generating test sentences that are grammatically correct and yield consistent translations.

---

Answer to RQ2: TransRepair has good precision (99%) for generating test sentences that are grammatically correct and yield consistent translations.
Table 1 shows the results of the reported inconsistent translations with different metric thresholds. From Table 1, we can see that bugs remain even for highly permissive consistency thresholds.

Table 1: Number of reported inconsistency bugs with different thresholds between 1.0 and 0.6. With a 1.0 threshold the translation is deemed buggy if there is any detected inconsistency. The lower the threshold, the more permissive is the criteria for deeming buggy. As can be seen, bugs remain even for highly permissive consistency thresholds (RQ2).

<table>
<thead>
<tr>
<th>Thresh.</th>
<th>1.0</th>
<th>0.9</th>
<th>0.8</th>
<th>0.7</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCS</td>
<td>2,053 (44%)</td>
<td>865 (18%)</td>
<td>342 (7%)</td>
<td>123 (3%)</td>
<td>57 (1%)</td>
</tr>
<tr>
<td>ED</td>
<td>2,053 (44%)</td>
<td>913 (19%)</td>
<td>401 (9%)</td>
<td>198 (4%)</td>
<td>101 (2%)</td>
</tr>
<tr>
<td>TF-idf</td>
<td>2,459 (52%)</td>
<td>548 (12%)</td>
<td>208 (4%)</td>
<td>71 (2%)</td>
<td>21 (0%)</td>
</tr>
<tr>
<td>BLEU</td>
<td>2,053 (44%)</td>
<td>1,621 (35%)</td>
<td>911 (19%)</td>
<td>510 (11%)</td>
<td>253 (5%)</td>
</tr>
<tr>
<td>LCS</td>
<td>2,213 (47%)</td>
<td>1,210 (26%)</td>
<td>634 (14%)</td>
<td>344 (7%)</td>
<td>184 (4%)</td>
</tr>
<tr>
<td>ED</td>
<td>2,213 (47%)</td>
<td>1,262 (27%)</td>
<td>700 (15%)</td>
<td>428 (9%)</td>
<td>267 (6%)</td>
</tr>
<tr>
<td>TF-idf</td>
<td>2,549 (54%)</td>
<td>851 (18%)</td>
<td>399 (9%)</td>
<td>188 (4%)</td>
<td>112 (2%)</td>
</tr>
<tr>
<td>BLEU</td>
<td>2,213 (47%)</td>
<td>1,857 (40%)</td>
<td>1,258 (27%)</td>
<td>788 (17%)</td>
<td>483 (10%)</td>
</tr>
</tbody>
</table>

Manual Inspected Inconsistency. In addition, we randomly sample 300 translations of the mutants. Two of them do not parse so we use the remaining 298 translations for analysis. For each mutant, the first two authors manually inspected its translation and the translation of the original sentence. An inconsistency is reported when any of the following criteria are met: Apart from the mutated substitute word, the two translations 1) have different meanings; 2) have different tones; 3) use different characters for proper nouns.

Manual inspection reveals 107 (36%) inconsistent translations for Google Translate, and 140 (47%) inconsistent translations for Transformer.

Correlation between Metrics and Manual Inspection. We compare metric scores and human consistency assessment results. We split the 298 human-labelled translations into two groups based on manual inspection. One is labelled as consistent translations, the other is labelled as inconsistent translations. We then check the metric value scores in each group.

Figure 4 shows the results. The points on the left/right of the dotted vertical line depict the metric values for the translations manually labelled as consistent/inconsistent. We observe that most points (82.2%) in the left section have a score of 1.0. The left section generally has higher score values (with a mean value of 0.99) than the right part (with a mean value of 0.86). These observations indicate that the metric values and manual inspection tend to agree on translation inconsistency. It is worth noting that Figure 4 also shows some low metric values on the left section and high metric values on the right section, which indicates the presence of false positives and false negatives of using metrics to assess consistency. We analyse false positives and false negatives in more detail below:

Threshold Learning. Metric scores are continuous values. To automatically report inconsistency, we need to set a threshold for each metric. In this paper, we choose a threshold that lets metric-value judgement best resemble manual inspection results. To do this, we randomly sampled another 100 translations from Google Translate and manually label them as consistent or not. We then used these 100 labels to choose a threshold (from 0.8 to 1.0 with a step of 0.01) with the largest F-measure score for each similarity metric. The best thresholds for LCS, ED, tf-idf, and BLEU identified in this way are 0.963, 0.963, 0.999, 0.906 with F-measure scores 0.81, 0.82, 0.79, 0.82, respectively. When a metric value falls below the so-identified threshold, our approach will report an inconsistency bug.

To know how well the chosen thresholds capture the boundary between consistency and inconsistency, we test the thresholds using the 298 previously sampled translations, on Google Translate and Transformer respectively. The results are shown in Table 2. A false positive (FP in the table) means the threshold judges a translation as inconsistent but manual inspection is consistent. A false negative (FN in the table) means the threshold judges a translation as consistent but manual inspection is inconsistent. From the table,
the proportion of false positives and false negatives are all below 10%, which we deem to be acceptable.

After a manual check of FPs and FNs, we found that an FN may happen when there is a minor character difference, but with a different meaning or tone. For example, in our results, one mutant translation has an extra er (means "But") which does not exist in the original translation. Manual inspection deemed this to be inconsistent, while metrics did not. An FP may happen when there are many different words in the two translations, but they share the same meaning. For example, Chinese phrases shang wei and hai mei you both mean "not yet", but the characters that express each phrase are totally different.

The harm that an FP in the testing process may bring lies in the possibility that our approach may make the translation worse. Section 4.3.2 explores this possibility.

Table 2: Precision/recall for inconsistency revealing (RQ2)

<table>
<thead>
<tr>
<th>Metric</th>
<th>TN</th>
<th>FN</th>
<th>FP</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>F-meas.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCS</td>
<td>169 (57%)</td>
<td>16 (5%)</td>
<td>22 (7%)</td>
<td>91 (31%)</td>
<td>0.81</td>
<td>0.85</td>
<td>0.83</td>
</tr>
<tr>
<td>ED</td>
<td>169 (57%)</td>
<td>16 (5%)</td>
<td>22 (7%)</td>
<td>91 (31%)</td>
<td>0.81</td>
<td>0.85</td>
<td>0.83</td>
</tr>
<tr>
<td>tf-idf</td>
<td>162 (54%)</td>
<td>12 (4%)</td>
<td>29 (10%)</td>
<td>95 (32%)</td>
<td>0.77</td>
<td>0.89</td>
<td>0.82</td>
</tr>
<tr>
<td>BLEU</td>
<td>173 (57%)</td>
<td>20 (7%)</td>
<td>20 (7%)</td>
<td>87 (29%)</td>
<td>0.81</td>
<td>0.81</td>
<td>0.81</td>
</tr>
</tbody>
</table>

| Transform | LCS     | 142 (48%) | 22 (7%) | 16 (5%) | 118 (40%) | 0.88 | 0.84 | 0.86 |
|           | ED      | 142 (48%) | 21 (7%) | 16 (5%) | 119 (40%) | 0.88 | 0.85 | 0.87 |
|           | tf-idf  | 141 (47%) | 11 (4%) | 17 (5%) | 129 (43%) | 0.88 | 0.92 | 0.90 |
|           | BLEU    | 147 (49%) | 23 (8%) | 11 (4%) | 117 (38%) | 0.91 | 0.84 | 0.87 |

Table 3: Number and proportion of repaired bugs (RQ3).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Probability</th>
<th>Cross-reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCS</td>
<td>–</td>
<td>493 (28%)</td>
</tr>
<tr>
<td>ED</td>
<td>–</td>
<td>503 (29%)</td>
</tr>
<tr>
<td>tf-idf</td>
<td>–</td>
<td>478 (25%)</td>
</tr>
<tr>
<td>BLEU</td>
<td>–</td>
<td>484 (29%)</td>
</tr>
<tr>
<td>Transformer</td>
<td>LCS</td>
<td>583 (30%)</td>
</tr>
<tr>
<td></td>
<td>ED</td>
<td>581 (30%)</td>
</tr>
<tr>
<td></td>
<td>tf-idf</td>
<td>640 (30%)</td>
</tr>
<tr>
<td></td>
<td>BLEU</td>
<td>565 (30%)</td>
</tr>
</tbody>
</table>

Table 3 shows the results, where each cell contains the number and proportion of inconsistency bugs that the repair approach repairs. The Column "Probability" represents the results for probability-referenced (grey-box repair); the Columns "Cross-reference" represents the results for cross-referencing (black-box repair). For Google Translate, since the grey-box approach is not applicable, we only present the results for the black-box approach.

From the table, TransRepair reduces, on average, 28% of bugs for the black-box approach in Google Translate. For the Transformer model, we can see the grey-box approach repairs 30% of bugs, the black-box one repairs 19% to 20% of bugs. These experimental results show that the grey-box and black-box approaches are effective at repairing inconsistency bugs.

4.3.2 Improvement Assessed by Human. Program repair studies typically include a manual assessment process [24, 39, 51] in order to validate their findings. Following a similar practice, the first two authors (manually) checked the repaired results of the previously labelled 298 sampled translations. The goal was to check whether the changes in translations patched by our repair approach improve translation consistency. Since our repair may also improve translation acceptability, we also check whether the changes bring any translation acceptability improvement. For cross-reference based repair, we manually assessed only the LCS metric since our (previous) results showed similar results among the other metrics.

Among the 298 sentence pairs, 113/136 of them are reported as having translation inconsistency on Google Translate/Transformer. Our repair thus targets all these sentence pairs, including the translations of both the original sentence and the mutant. The probability-based repair approach finally changed 58 (out of 136) pairs for Transformer; The cross-reference-based repair approach finally changed 39/27 (out of 113/136) pairs for Google Translate/Transformer.

For the translation pairs that have been modified by TransRepair, the first two authors then manually compared two dimensions: 1) the translation consistency before and after repair; 2) the acceptability of the translations (for both the original sentence and the mutant) before and after repair. For each dimension, the authors gave labels of “Improved”, “Unchanged”, or “Decreased”, considering both adequacy and fluency (see explanations of these two terms in Section 2.3).

Table 4 shows the results. The first four rows are for Google Translate. The remaining rows are for Transformer. The rows with “overall” show results of translation acceptability improvement.

Answers to RQ2: The metrics have an F-measure of 0.82/0.88 when detecting inconsistency bugs for Google Translate/Transformer. Both metrics and manual inspection reveal that Google Translate has approximately 36% inconsistent translations on TransRepair test inputs.

4.3 Bug-repair Ability (RQ3)

4.3.1 Improvement Assessed by Metrics. We apply our repair approaches to all the inconsistent translations, and check how many translations can be repaired with our approach. For each sentence, we generate 16 mutants for repair (we study the influence of the number of mutants for repair in Section 5).

*We removed those 100 translations used for threshold learning from our test set and used the remaining 4,592 inputs to conduct testing.*
### Table 4: Improvement Based on Manual Inspection (RQ3)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Improved</th>
<th>Unchanged</th>
<th>Decreased</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation consistency</td>
<td>33 (85%)</td>
<td>4 (10%)</td>
<td>2 (5%)</td>
</tr>
<tr>
<td>Translation acceptability: overall</td>
<td>22 (28%)</td>
<td>48 (62%)</td>
<td>8 (10%)</td>
</tr>
<tr>
<td>Translation acceptability: original</td>
<td>10 (26%)</td>
<td>23 (59%)</td>
<td>6 (15%)</td>
</tr>
<tr>
<td>Translation acceptability: mutant</td>
<td>12 (31%)</td>
<td>25 (64%)</td>
<td>2 (5%)</td>
</tr>
<tr>
<td>Translation consistency</td>
<td>24 (89%)</td>
<td>3 (11%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Translation acceptability: overall</td>
<td>15 (28%)</td>
<td>37 (69%)</td>
<td>2 (4%)</td>
</tr>
<tr>
<td>Translation acceptability: original</td>
<td>7 (26%)</td>
<td>19 (70%)</td>
<td>1 (4%)</td>
</tr>
<tr>
<td>Translation acceptability: mutant</td>
<td>8 (30%)</td>
<td>18 (67%)</td>
<td>1 (4%)</td>
</tr>
<tr>
<td>Translation consistency</td>
<td>51 (88%)</td>
<td>6 (10%)</td>
<td>1 (2%)</td>
</tr>
<tr>
<td>Translation acceptability: overall</td>
<td>30 (26%)</td>
<td>76 (66%)</td>
<td>10 (9%)</td>
</tr>
<tr>
<td>Translation acceptability: original</td>
<td>15 (26%)</td>
<td>36 (62%)</td>
<td>7 (12%)</td>
</tr>
<tr>
<td>Translation acceptability: mutant</td>
<td>15 (26%)</td>
<td>40 (69%)</td>
<td>3 (5%)</td>
</tr>
</tbody>
</table>

**Answers to RQ3: Black-box repair reduces on average 28%/19% bugs for Google Translate/Transformer. Grey-box repair reduces on average 30% bugs for Transformer. Human inspection indicates that the repaired translations improve consistency in 87% of the cases (reducing it in 2%), and have better translation acceptability in 27% of the cases (worse in 8%).**

### Effectiveness and Efficiency of Data Augmentation

Previous work has adopted data augmentation to improve the robustness of machine learning models [5, 36]. In our work, concerning translators whose source code is known, training data augmentation is also a candidate solution to increase translation consistency.

To investigate this option, we designed experiments to study whether adding more training data would yield better translation consistency. We controlled the training data size and used 10%, 20%, ..., and 90% of the original training data to build Transformer respectively. Figure 5 shows the results. When the size of the training data ratio is between 0.7 and 1.0, we did not observe a decreasing trend. This indicates that augmenting training data may have limited effectiveness in improving translation inconsistency.

**Figure 5: Ratio of inconsistency bugs with different training-data size for Transformer.**

Data augmentation needs model retraining. We found that using 100% training data to train the translator model took as much as 19 hours under our current experimental configuration (see more details in Section 3). In practice, data collection, labelling, and processing also needs time. All together, we find little evidence that augmentation is a complete solution to this translation repair problem.

Compared with model retraining approaches like training data augmentation, TransRepair has the following advantages: 1) TransRepair require neither the source code nor training data, and is either completely black-box or requires only predictive probability (grey-box); 2) TransRepair can have lower repair cost since it does not need additional data and does not require model retraining; 3) TransRepair is more flexible, because it enables the repair of a specific bug without touching other well-formed translations.

### Efficiency of TransRepair

The cost of TransRepair includes both testing and repair. Under our current experimental configuration (see more details in Section 3), the mean time for testing is 0.97s per sentence; the mean time for repair is 2.68s per sentence for the probability-based approach, and 2.93s per sentence for the cross-reference-based approach. Thus, with current experimental configuration, when using TransRepair to optimise the real-time online machine translation, for a sentence that is deemed non-buggy, it would take less than 1 second for the end user to receive a final translation. For a sentence that is deemed buggy, it would take less than 4 seconds (testing and repair) to get a final translation.
Table 5: Examples of Repaired Translations.

<table>
<thead>
<tr>
<th>Input</th>
<th>Original translation</th>
<th>Repaired translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female students do <strong>good</strong> research in computer science.</td>
<td>nüxuesheng zai jisuanji kexue fangmian zuole hen-duo yanjiu</td>
<td>nüxueshengzai jisuanji kexue fangmian zuole hen-haode yanjiu</td>
</tr>
<tr>
<td>If you need help, you can enjoy timely services by pressing a nearby one of the 41 call buttons in the station.</td>
<td>ruguo ni xuyao bangzhu, ni keyi tonggao an fujin de 41 ge hujiao annin xiangzhong jishi de fuwu. [Bug: &quot;one of&quot; is not translated.]</td>
<td>ruguo ni xuyao bangzhu, ni keyi tonggao an fujin de 41 ge hujiao annin zhong de yige lai xiangzhong jishi de fuwu.</td>
</tr>
</tbody>
</table>

Table 6: Number of detected and repaired bugs with different number of mutants.

<table>
<thead>
<tr>
<th>Metric</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCS</td>
<td>35</td>
<td>1345</td>
<td>1917</td>
<td>490</td>
<td>551</td>
</tr>
<tr>
<td>ED</td>
<td>536</td>
<td>1349</td>
<td>1923</td>
<td>488</td>
<td>549</td>
</tr>
<tr>
<td>tf-idf</td>
<td>583</td>
<td>1467</td>
<td>2102</td>
<td>534</td>
<td>600</td>
</tr>
<tr>
<td>BLEU</td>
<td>513</td>
<td>1300</td>
<td>1857</td>
<td>483</td>
<td>538</td>
</tr>
</tbody>
</table>

**Influence of Mutant Number.** We generate mutants during both inconsistency testing and repair. For the test/repair process, our default configuration generates at most 5/16 mutants for each sentence. To investigate how the number of mutants affects the testing and repair performance, we repeat our experiments with 1 or 3 mutants for test generation, and with 4 or 8 mutants for repair. We then compare the number of revealed inconsistency bugs and the number of bugs that our approaches repair. For repair, we only present results of grey-box repair approach in the paper due to space reason, as shown by Table 6. The full results are available on our homepage [43]. We observe that during testing, using more mutants helps to reveal more inconsistency bugs. It is the same for repair, but using 4 mutants during repair also has an acceptable effectiveness.

**Application Scenario.** TransRepair can be applied end to end. Given a translation input and a machine translator, our approach will automatically test and repair the translation output, and give a new translation output to the end user.

6 RELATED WORK

Software testing research has typically targeted traditional (non-machine-learning-based) software systems. However, the recent rise in the real-world importance of machine learning has resulted in a concomitant rise in the level of research activity in software testing for machine learning [56]. At the same time, software repair concepts and techniques [23, 25, 28] remain relatively under explored for machine learning systems. In this section, we review the relationship of our proposed machine translation testing and repair with previous work on testing and repair machine translation systems, which mainly focus on translation robustness.

**Translation Robustness Testing.** To test translation robustness, researchers have explored how perturbations on the test inputs affect translations. Heigold et al. [20] studied three types of character-level noisy test inputs that are generated via character swapping, word scrambling, and character flipping. They found that machine translation systems are very sensitive to slightly perturbed sentences that do not pose a challenge to humans. Belinkov and Bisk [2] had a similar conclusion, not only on synthetic noise, but also on natural noise (naturally occurring errors like typos and misspellings). To have more diverse test cases for robustness testing, Zhao et al. [58] used Generative Adversarial Networks (GANs) [13]. They projected the input sentence to a latent space, which is then used for searching for test sentences close to the input.

These work targets robustness testing. The test inputs are synthetic errors or naturally-occurring errors. The test oracles are usually BLEU scores, which are bounded with human oracles. Our approach targets translation consistency, and we generate consistent test inputs by context-similar word replacement, instead of involving errors. Our approach also does not require human oracles during testing.

Sun and Zhou [42] proposed metamorphic relations for machine translation. There are two major differences between our work and theirs: 1) their work concerns testing only; we also repair; 2) their test input generation merely replaces human names before “likes” or “hates” and brands after them; our approach is comparatively more exhaustive.

**Translation Robustness Improvement.** To improve translation robustness, previous work relies largely on data augmentation, i.e., to add noisy data into the training data and to retrain the model. Some work used model-independent data generation (also called black-box data generation). Heigold et al. [20], Belinkov and Bisk [2], and Sperber et al. [41] used synthetic noise to retain the model. Karpukhin et al. [26] evaluated the impact of percentages of synthetic noise to the training set.

Some work uses model-dependent data generation (white-box data generation). Ebrahimi et al. [9] introduced an approach that relies on an atomic flip operation. This operation generates tests by swapping characters based on the gradients of the input vectors. Cheng et al. [4] proposed a gradient-based method to generate adversarial sentences by conducting word replacement.

There is also work on improving robustness via optimizing the learning algorithms. Belinkov and Bisk [2] proposed to use
a structure-invariant representation for synthetic noise in the network. They find that a character CNN representation is more robust than others. Cheng et al. [5] introduced stability training by adding a new component for discriminating the noise in the training set. This component reduces the impact of the noise, and yields more stable translations when making synonymous perturbations.

These previous approaches target overall robustness improvement for all translations, rather than fixing specific mistranslations.

7 CONCLUSION

In this paper, we presented TransRepair, the first approach that automatically tests and improves context-similar translation consistency. TransRepair takes a sentence and applies a context-similar mutation to generate slightly altered (mutated) sentences, to be used to test machine translation systems. Testing is performed by translating and comparing the original with the mutated sentences. To judge consistency, TransRepair calculates the similarity of the translation subsequences. When context-similar mutations yield above-threshold disruption to the translation of the unchanged part, TransRepair deems this to be a potential bug. In addition to testing, TransRepair also automatically repairs inconsistencies in a blackbox or grey-box manner, which post-processes the translations with reference to the translations of mutated sentences.

ACKNOWLEDGEMENT

Zeyu Sun and Lu Zhang are supported by the National Key Research and Development Program of China under Grant No. 2017YFB1001803. Jie M. Zhang and Mark Harman are supported by the ERC advanced grant with No. 741278. Mike Papadakis is supported by the Luxembourg National Research Funds (FNR) C17/IS/11686509/CODEMATES.

REFERENCES
