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.

English	Chinese (Google Translation)	Notes
Men do good research in computer science.	Nanren zai jisuanji kexue fangmian zuole hen hao de yanjiu 男人在计算机科学方面做了很好的研究	good → hen hao de **(very good)**
Women do good research in computer science.	Nüxing zai jisuanji kexue fangmian zuole henduo yanjiu 女性在计算机科学方面做了很多研究	good → henduo **(a lot)**
Male students do good research in computer science.	Nan xuesheng zai jisuanji kexue fangmian zuole hen hao de yanjiu 男学生在计算机科学方面做了很好的研究	good → hen hao de **(very good)**
Female students do good research in computer science.	Nü xuesheng zai jisuanji kexue fangmian zuole henduo yanjiu 女学生在计算机科学方面做了很多研究	good → henduo **(a lot)**

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-equal transformations,

*Corresponding and co-first author.
HCST: High Confidence Software Technologies.
This paper is accepted by the technical track of ICSE 2020

[1]The 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.
[2]Similar 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[3] 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 metamorphic 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 $\boldsymbol{v}_x$ denotes the vector of the word x.

$$\text{sim}(w_1, w_2) = \frac{\boldsymbol{v}_{w_1} \boldsymbol{v}_{w_2}}{|\boldsymbol{v}_{w_1}||\boldsymbol{v}_{w_2}|} \quad (1)$$

[3]We use "acceptability" to capture the property that a translation meets human assessment of a reasonable (aka acceptable) translation

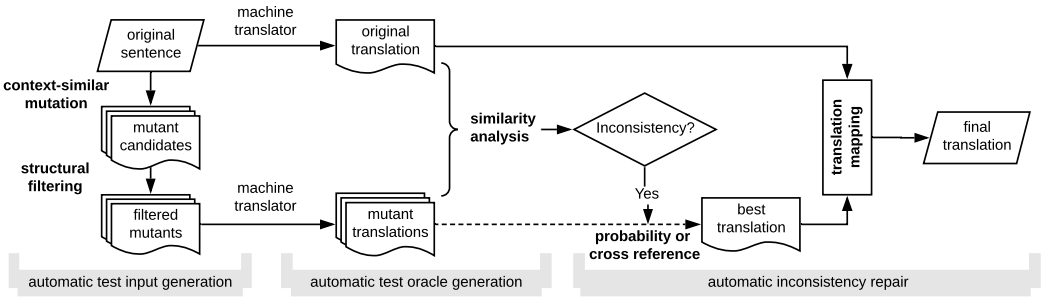

Figure 2: Overview of how TransRepair tests and repairs machine translation inconsistencies.

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 $B_{s'}$[4]. 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_o/T_m (Line 6 and Line 10).

For each element in set T_o, we compute its similarity[5] with each element in the set T_m (Line 15). Thus, we get $|T_o| * |T_m|$ similarity scores, where $|T_o|$ and $|T_m|$ is the size of T_o 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

[4]Long slices are unlikely to correspond to the mutated word, we thus only keep slices that are no longer than 5 words.
[5]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 $t(s')$

$B_s, B_{s'}$ = Wdiff$(t(s), t(s'))$
$T_o = \{t(s)\}$
$T_m = \{t(s')\}$
for *each subsequence* $b_s \in B_s$ **do**
  r = DeleteSub$(t(s), b_s)$
  $T_o = T_o \cup \{r\}$
end
for *each subsequence* $b_{s'} \in B_{s'}$ **do**
  r' = DeleteSub$(t(s'), b_{s'})$
  $T_m = T_m \cup \{r'\}$
end
ConScore = -1
for *each sentence* $a \in T_o$ **do**
  for *each sentence* $b \in T_m$ **do**
    Sim = ComputeSimilarity(a, b)
    ConScore = Max$(ConScore, Sim)$
  end
end
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

Algorithm 2: Process of automatic repair

Data: s: a sentence input; $t(s)$: translation of s; $s_1, s_2, ..., s_n$: mutants of s; $t(s_1), t(s_2), ..., t(s_n)$: translations of the mutants;
Result: NewTrans: repaired translation for s

1 $T = \{(s, t(s)), (s_1, t(s_1)), (s_2, t(s_2)), ..., (s_n, t(s_n))\}$
2 OrderedList = Rank(T)
3 $a(s)$ = wordAlignment$(s, t(s))$
4 NewTrans = $t(s)$
5 **for** *each sentence and its translation* $s_r, t(s_r) \in$ OrderedList **do**
6 **if** $s_r == s$ **then**
7 break
8 **end**
9 $a(s_r)$ = wordAlignment$(s_r, t(s_r))$
10 w_i, w_i^r = getReplacedWord(s, s_r)
11 $t(w_i)$ = getTranslatedWord$(w_i, a(s))$
12 $t(w_i^r)$ = getTranslatedWord$(w_i^r, a(s_r))$
13 **if** isnumeric(w_i)! = isnumeric$(t(w_i))$ ***or*** isnumeric(w_i^r)! = isnumeric$(t(w_i^r))$ **then**
14 continue
15 **end**
16 $t^r(s_r)$ = mapBack(t(s), t(s_r), s, s_r, a(s), a(s_r))
17 **if** ***not*** (isnumeric(w_i) ***and*** isnumeric(w_i^r)) **then**
18 **if** structure$(t^r(s_r))$! = structure$(t(s_r))$ **then**
19 continue
20 **end**
21 **end**
22 **if** isTest(s) **then**
23 $s_o, t(s_o)$ = getRepairedResult(s)
24 **if** *not* $isConsistent(t(s_o), t^r(s_r))$ **then**
25 continue
26 **end**
27 **end**
28 NewTrans = $t^r(s_r)$
29 break
30 **end**
31 **return** NewTrans

alignment to s_1 and $t(s_1)$ as to s and $t(s)$. The variables w_i, w_i^r denote the replaced words in s, s_r and we get the translated words $t(w_i)$, $t(w_i^r)$ through the alignment (Lines 9-12).

Word alignment is not 100% accurate. If we directly map the translation by replacing $t(w_i^r)$ 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^r and $t(w_i)$, $t(w_i^r)$ 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.

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

$$M_{LCS} = \frac{len(LCS(t_1, t_2))}{Max(len(t_1), len(t_2))} \tag{2}$$

In this formula, LCS is a function that calculates a longest common subsequence [21] between t_1 and t_2 that appear in the same relative order. For example, an LCS for input Sequences "ABCDGH" and "AEDFHR" is "ADH" with a length of 3.

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].

$$M_{ED} = 1 - \frac{ED(t_1, t_2)}{Max(len(t_1), len(t_2))} \tag{3}$$

In this formula, ED is a function that calculates the edit distance between t_1 and t_2.

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.

$$w_{idf} = \log((|C| + 1)/(f_w + 1)) \tag{4}$$

We then represent each sentence with the bag-of-words model [57], which is a simplified representation commonly used in natural language processing. In this model, the grammar and the word order is disregarded, only keeping multiplicity (i.e., "A B C A" is represented as "A":2, "B":1, "C":1, namely [2, 1, 1] in vector). Each dimension of the vector is multiplied with its weight w_{idf}. We calculate the cosine similarity (Equation 1) of the weighted vectors of t_1 and t_2 as their final tf-idf-based consistency score.

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 subsequences between sentences and computes a precision p_n (which is called modified n-gram precision [33], where n means the subsequence 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 overaly short translations), which is shown by Formula 5. c denotes the length of $t(s_i)$ and r is the length of $t(s)$.

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{5}$$

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.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right). \tag{6}$$

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 Tensor2Tensor 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. set[6].

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.

[6]The Chinese sentences in our experiments are in the form of characters.

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.

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

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.

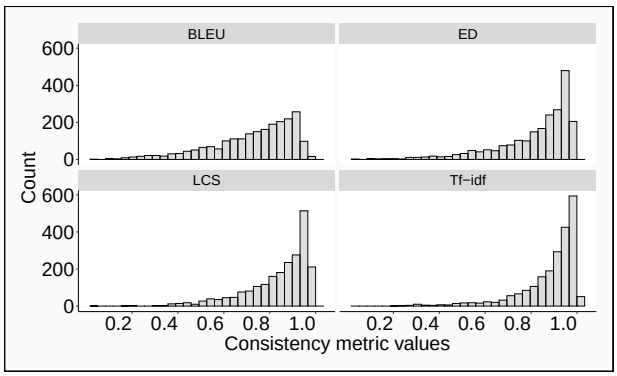

Figure 3: Histogram of metric scores. A large number of mutant translations have similarity scores lower than one, indicating many inconsistent translations (RQ2).

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).

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

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[7].

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.

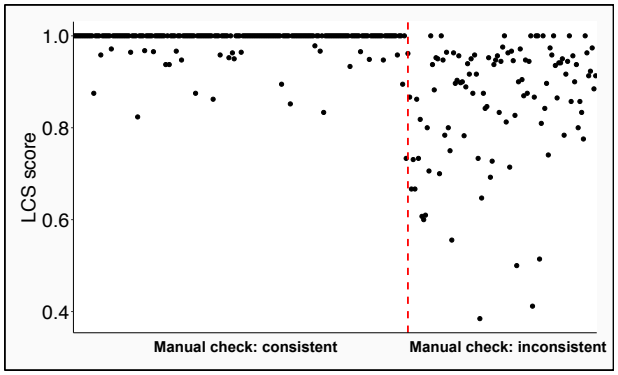

Figure 4: Comparison of metric scores and manual inspection of translation consistency. There is a good agreement between metric score values and human assessment (RQ2).

Figure 4 shows the results[8]. 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,

[7]The Cohen's Kappa is 0.96/0.95 for Google Translate/Transformer, indicating that the inspection results are highly consistent.

[8]This figure shows only the LCS metric. Full results are on our homepage [43].

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)

	Metric	TN	FN	FP	TP	Precision	Recall	F-meas.
GT	LCS	169 (57%)	16 (5%)	22 (7%)	91 (31%)	0.81	0.85	0.83
	ED	169 (57%)	16 (5%)	22 (7%)	91 (31%)	0.81	0.85	0.83
	tf-idf	162 (54%)	12 (4%)	29 (10%)	95 (32%)	0.77	0.89	0.82
	BLEU	171 (57%)	20 (7%)	20 (7%)	87 (29%)	0.81	0.81	0.81
Transformer	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 (39%)	0.91	0.84	0.87

Overall Number of Inconsistency Issues. After identifying the thresholds, we apply them to the translations of the 4,592 generated inputs[9], and check how many of them fall below the threshold. It turns out that for Transformer, the inconsistent translation results are LCS: 1,917 (42%); ED: 1,923 (42%); tf-idf: 2,102 (46%); BLEU: 1,857 (40%). Thus, overall, about two fifths of the translations fall below our chosen consistency thresholds. For Google Translate, the inconsistent translation results are LCS: 1,708 (37%); ED: 1,716 (37%); tf-idf: 1,875 (41%); BLEU: 1,644 (36%). This also shows that Google Translate is slightly better than Transformer with respect to consistency.

> 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).

[9]We removed those 100 translations used for threshold learning from our test set and used the remaining 4,592 inputs to conduct testing.

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

	Metric	Probability	Cross-reference
GT	LCS	–	493 (28%)
	ED	–	503 (29%)
	tf-idf	–	478 (25%)
	BLEU	–	484 (29%)
Transformer	LCS	583 (30%)	374 (19%)
	ED	581 (30%)	389 (20%)
	tf-idf	640 (30%)	400 (19%)
	BLEU	565 (30%)	380 (20%)

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-reference (grey-box repair); the Columns "Cross-reference" represents the results for cross-reference (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 by metrics. 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)[10].

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

[10]The mean Kappa score is for labelling translation consistency between the two human labels, 0.97 for labelling the translation of the original sentence, and 0.81 for labelling the translation of the mutant sentence.

Table 4: Improvement Based on Manual Inspection (RQ3)

	Aspect	Improved	Unchanged	Decreased
GT.LCS	**Translation consistency**	33 (85%)	4 (10%)	2 (5%)
	Translation acceptability: overall	22 (28%)	48 (62%)	8 (10%)
	Translation acceptability: original	10 (26%)	23 (59%)	6 (15%)
	Translation acceptability: mutant	12 (31%)	25 (64%)	2 (5%)
Trans.LCS	**Translation consistency**	24 (89%)	3 (11%)	0 (0%)
	Translation acceptability: overall	15 (28%)	37 (69%)	2 (4%)
	Translation acceptability: original	7 (26%)	19 (70%)	1 (4%)
	Translation acceptability: mutant	8 (30%)	18 (67%)	1 (4%)
Trans.Prob	**Translation consistency**	51 (88%)	6 (10%)	1 (2%)
	Translation acceptability: overall	30 (26%)	76 (66%)	10 (9%)
	Translation acceptability: original	15 (26%)	36 (62%)	7 (12%)
	Translation acceptability: mutant	15 (26%)	40 (69%)	3 (5%)

among the translations for both original sentences and mutant sentences. The rows with "original"/"mutant" show the translation repair results for original/mutant sentences.

We observe that TransRepair has a good effectiveness in improving translation consistency. For example, on average, 87% translation pairs improve the consistency for Google Translate and Transformer, while all together we observe only 3 translation consistency decreases. We checked these decreased-consistency repairs and found that, for one case, the original sentence translation has been improved but not for the mutant translation, and thus after repair, the improved translation of the original sentence does not match well with the unimproved translation of the mutant. The remaining two cases arise because the repairs of the original sentences decreased, while our approach did not touch the mutant translations.

The main purpose of our repair approach is to improve translation consistency. Translation acceptability improvement is a "bonus" of our approach. From Table 4, perhaps to our surprise, the repair approach improves the translation acceptability for around one fourth (27%) repairs. There are 8% repairs with decreased acceptability. Based on our manual inspection, the reason for decreased acceptability is that occasionally, the repair approach may trade quality for consistency.

> 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%).

5 EXTENDED ANALYSIS AND DISCUSSION

This section provides further details and analysis.

Example of Repaired Translations. Table 5 gives some examples of our improvement of mistranslation. The first column is the translation input; the second column shows the original translation output (converted to Pinyin), where words in italic explain the mistranslated parts; the last column shows our improved translation.

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.

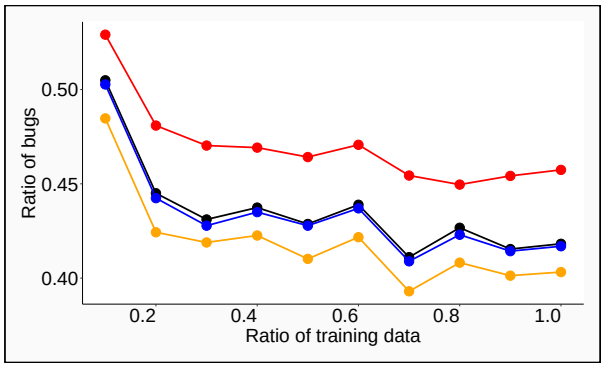

Figure 5: Ratio of inconsistency bugs with different training-data size for Transformer.

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.

Input	Original translation	Repaired translation
Female students do **good** research in computer science.	nüxuesheng zai jisuanji kexue fangmian zuole **henduo** yanjiu *[Bug: "good" → "a lot".]*	nüxueshengzai jisuanji kexue fangmian zuole **henhaode** yanjiu
If you need help, you can enjoy timely services by pressing a nearby **one of** the 41 call buttons in the station.	ruguo ni xuyao bangzhu, ni keyi tongguo an fujin de 41 ge hujiao anniu xiangshou jishi de fuwu. *[Bug: "one of" is not translated.]*	ruguo ni xuyao bangzhu, ni keyi tongguo an fujin de 41 ge hujiao anniu **zhong de yige** lai xiangshou jishi de fuwu.
Original Title : Canada Police Kill **IS** Supporters : Preparations for Homemade Bomb Downtown Attack Near the End.	yuanshi biaoti: jianada jingcha sha le **wo de** zhichizhe: wei hema zhizao baozha de zhunbei. *[Bug: "IS" → "my".]*	yuanshi biaoti: jianada jingcha shalu **IS** zhichizhe: wei hema zhizao zhadan de zhunbei.
Ban political campaigners and activists from handling **completed** postal votes and postal vote envelopes.	jinzhi zhengzhi jingsuanzhe he huodongfenzi chuli **wan** de youzheng xuanpiao he youzheng xuanpiao xifeng *[Bug: "completed" is mistranslated.]*	jinzhi zhengzhi jingsuanzhe he huodongfenzi chuli **yi wancheng** de youzheng xuanpiao he youzheng xuanpiao xifeng.

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

Metric	Mutant number for testing			Mutant number for repair		
	1	3	5	4	8	16
LCS	535	1,345	1,917	490	551	583
ED	536	1,349	1,923	488	549	581
tf-idf	583	1,467	2,102	534	600	640
BLEU	513	1,300	1,857	483	538	565

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 optimising 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 black-box 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

[1] Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, Ann Houston. 2013. OntoNotes. https://catalog.ldc.upenn.edu/LDC2013T19.

[2] Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *Proc. ICLR*.

[3] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases.* Technical Report.

[4] Yong Cheng, Lu Jiang, and Wolfgang Macherey. 2019. Robust Neural Machine Translation with Doubly Adversarial Inputs. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. 4324–4333. https://www.aclweb.org/anthology/P19-1425/

[5] Yong Cheng, Zhaopeng Tu, Fandong Meng, Junjie Zhai, and Yang Liu. 2018. Towards robust neural machine translation. *arXiv preprint arXiv:1805.06130* (2018).

[6] CWMT. 2018. The CWMT Dataset. http://nlp.nju.edu.cn/cwmt-wmt/.

[7] George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*. Morgan Kaufmann Publishers Inc., 138–145.

[8] David M Eberhard, Gary F Simons, and Charles D Fennig. 2019. Ethnologue: Languages of the world. (2019).

[9] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Melbourne, Australia, 31–36. https://doi.org/10.18653/v1/P18-2006

[10] Free Software Foundation. 2019. GNU Wdiff. https://www.gnu.org/software/wdiff/

[11] Carlo Giglio and Richard Caulk. 1965. Article 17 of the Treaty of Uccialli. *The Journal of African History* 6, 2 (1965), 221–231.

[12] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[14] Google. 2019. Google Translate. http://translate.google.com.

[15] Yvette Graham, Timothy Baldwin, Aaron Harwood, Alistair Moffat, and Justin Zobel. 2012. Measurement of progress in machine translation. In *Proceedings of the Australasian Language Technology Association Workshop 2012*. 70–78.

[16] Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2018. Search engine guided neural machine translation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[17] Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. 2019. Modeling Recurrence for Transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 1198–1207. https://doi.org/10.18653/v1/N19-1122

[18] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. 2018. Achieving Human Parity on Automatic Chinese to English News Translation. *CoRR* abs/1803.05567 (2018). arXiv:1803.05567 http://arxiv.org/abs/1803.05567

[19] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *24th International Symposium on High-Performance Computer Architecture (HPCA 2018), February 24-28, Vienna, Austria*.

[20] Georg Heigold, Stalin Varanasi, Günter Neumann, and Josef van Genabith. 2018. How Robust Are Character-Based Word Embeddings in Tagging and MT Against Wrod Scramlbing or Randdm Nouse?. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas, AMTA 2018, Boston, MA, USA, March 17-21, 2018 - Volume 1: Research Papers*. 68–80. https://aclanthology.info/papers/W18-1807/w18-1807

[21] James W Hunt and Thomas G Szymanski. 1977. A fast algorithm for computing longest common subsequences. *Commun. ACM* 20, 5 (1977), 350–353.

[22] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (September–October 2011), 649 – 678.

[23] Jiajun Jiang, Yingfei Xiong, and Xin Xia. 2019. A manual inspection of Defects4J bugs and its implications for automatic program repair. *Science China Information Sciences* 62, 10 (2019), 200102.

[24] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping Program Repair Space with Existing Patches and Similar Code. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*. ACM, New York, NY, USA, 298–309. https://doi.org/10.1145/3213846.3213871

[25] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping program repair space with existing patches and similar code. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 298–309.

[26] Vladimir Karpukhin, Omer Levy, Jacob Eisenstein, and Marjan Ghazvininejad. 2019. Training on Synthetic Noise Improves Robustness to Natural Noise in Machine Translation. *arXiv preprint arXiv:1902.01509* (2019).

[27] Huda Khayrallah and Philipp Koehn. 2018. On the impact of various types of noise on neural machine translation. *arXiv preprint arXiv:1805.12282* (2018).

[28] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2011. Genprog: A generic method for automatic software repair. *Ieee transactions on software engineering* 38, 1 (2011), 54–72.

[29] Yang Liu and Maosong Sun. 2015. Contrastive unsupervised word alignment with non-local features. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[30] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010

[31] M. Chris Mason. 2017. Strategic Insights: Lost in Translation. https://ssi.armywarcollege.edu/index.cfm/articles/Lost-In-Translation/2017/08/17

[32] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: an analysis and survey. In *Advances in Computers*. Vol. 112. Elsevier, 275–378.

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 311–318.

[34] Parmy Olson. 2018. The Algorithm That Helped Google Translate Become Sexist. https://www.forbes.com/sites/parmyolson/2018/02/15/the-algorithm-that-helped-google-translate-become-sexist/#224101cb7daa.

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

[36] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 856–865.

[37] Eric Sven Ristad and Peter N Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 5 (1998), 522–532.

[38] Robert Parker, David Graff, Junbo Kong, Ke Chen, Kazuaki Maeda. 2011. English Gigaword Fifth Edition. https://catalog.ldc.upenn.edu/LDC2011T07.

[39] Ripon K. Saha, Yingjun Lyu, Hiroaki Yoshida, and Mukul R. Prasad. 2017. ELIXIR: Effective Object Oriented Program Repair. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 648–659. http://dl.acm.org/citation.cfm?id=3155562.3155643

[40] SpaCy. 2019. SpaCy. https://spacy.io/.

[41] Matthias Sperber, Jan Niehues, and Alex Waibel. 2017. Toward robust neural machine translation for noisy input sequences. In *International Workshop on Spoken Language Translation (IWSLT)*.

[42] Liqun Sun and Zhi Quan Zhou. 2018. Metamorphic testing for machine translations: MT4MT. In *2018 25th Australasian Software Engineering Conference (ASWEC)*. IEEE, 96–100.

[43] Zeyu Sun. 2019. TransRepair Homepage. https://github.com/zysszy/TransRepair.

[44] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. The Penn treebank: an overview. In *Treebanks*. Springer, 5–22.

[45] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2Tensor for Neural Machine Translation. *CoRR* abs/1803.07416 (2018). http://arxiv.org/abs/1803.07416

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 6000–6010.

[47] VoiceBoxer. 2016. WHAT ABOUT ENGLISH IN CHINA? http://voiceboxer.com/english-in-china/.

[48] Rining Wei and Jinzhi Su. 2012. The statistics of English in China: An analysis of the best available data from government sources. *English Today* 28, 3 (2012), 10–14.

[49] Wikipedia. 2014. Wikipedia. https://dumps.wikimedia.org/.

[50] WMT. 2018. News-Commentary. http://data.statmt.org/wmt18/translation-task/.

[51] Qi Xin and Steven P. Reiss. 2017. Leveraging Syntax-related Code for Automated Program Repair. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 660–670. http://dl.acm.org/citation.cfm?id=3155562.3155644

[52] Jie Zhang, Junjie Chen, Dan Hao, Yingfei Xiong, Bing Xie, Lu Zhang, and Hong Mei. 2014. Search-based inference of polynomial metamorphic relations. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 701–712.

[53] Jingyi Zhang, Masao Utiyama, Eiichro Sumita, Graham Neubig, and Satoshi Nakamura. 2018. Guiding neural machine translation with retrieved translation pieces. *arXiv preprint arXiv:1804.02559* (2018).

[54] Jie Zhang, Lingming Zhang, Mark Harman, Dan Hao, Yue Jia, and Lu Zhang. 2018. Predictive mutation testing. *IEEE Transactions on Software Engineering* 45, 9 (2018), 898–918.

[55] Jie Zhang, Muyao Zhu, Dan Hao, and Lu Zhang. 2014. An empirical study on the scalability of selective mutation testing. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 277–287.

[56] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2019. Machine Learning Testing: Survey, Landscapes and Horizons. *arXiv preprint arXiv:1906.10742* (2019).

[57] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1, 1 (01 Dec 2010), 43–52. https://doi.org/10.1007/s13042-010-0001-0

[58] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2017. Generating Natural Adversarial Examples. *CoRR* abs/1710.11342 (2017). arXiv:1710.11342 http://arxiv.org/abs/1710.11342

[59] Michał Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. 2016. The united nations parallel corpus v1. 0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. 3530–3534.