Structure-Invariant Testing for Machine Translation

Pinjia He
Department of Computer Science
ETH Zurich
Switzerland
pinjia.he@inf.ethz.ch

Clara Meister
Department of Computer Science
ETH Zurich
Switzerland
clara.meister@inf.ethz.ch

Zhendong Su
Department of Computer Science
ETH Zurich
Switzerland
zhendong.su@inf.ethz.ch

ABSTRACT

In recent years, machine translation software has increasingly been integrated into our daily lives. People routinely use machine translation for various applications, such as describing symptoms to a foreign doctor and reading political news in a foreign language. However, the complexity and intractability of neural machine translation (NMT) models that power modern machine translation make the robustness of these systems difficult to even assess, much less guarantee. Machine translation systems can return inferior results that lead to misunderstanding, medical misdiagnoses, threats to personal safety, or political conflicts. Despite its apparent importance, validating the robustness of machine translation systems is very difficult and has, therefore, been much under-explored.

To tackle this challenge, we introduce *structure-invariant testing (SIT)*, a novel metamorphic testing approach for validating machine translation software. Our key insight is that the translation results of "similar" source sentences should typically exhibit similar sentence structures. Specifically, SIT (1) generates similar source sentences by substituting one word in a given sentence with semantically similar, syntactically equivalent words; (2) represents sentence structure by syntax parse trees (obtained via constituency or dependency parsing); (3) reports sentence pairs whose structures differ quantitatively by more than some threshold. To evaluate SIT, we use it to test Google Translate and Bing Microsoft Translator with 200 source sentences as input, which led to 64 and 70 buggy issues with 69.5% and 70% top-1 accuracy, respectively. The translation errors are diverse, including under-translation, over-translation, incorrect modification, word/phrase mistranslation, and unclear logic.

KEYWORDS

Metamorphic testing, Machine translation, Structural invariance

1 INTRODUCTION

Machine translation software has seen rapid growth in the last decade; users now rely on machine translation for a variety of applications, such as signing lease agreements when studying abroad, describing symptoms to a foreign doctor, and reading political news in a foreign language. In 2016, Google Translate, the most widely-used online translation service, attracted more than 500 million users and translated more than 100 billion words per day [81]. On top of this, machine translation services are also embedded into various software applications, such as Facebook [25] and Twitter [82].

The advances in machine translation that are responsible for such growth can largely be attributed to neural machine translation (NMT) models, which have become the core component of many machine translation systems. As reported by research from Google [86] and Microsoft [32], state-of-the-art NMT models are approaching human-level performance in terms of accuracy, i.e., BLEU [67]. These recent breakthroughs have led users to start relying on machine translation software (e.g., Google Translate [30] and Bing Microsoft Translator [5]) in their daily lives.

However, NMT models are not as reliable as many may believe. Recently, sub-optimal and incorrect outputs have been found in various software systems with neural networks as their core components. Typical examples include autonomous cars [23, 68, 79], sentiment analysis tools [2, 36, 46], and speech recognition services [6, 71]. These recent research efforts show that neural networks can easily return inferior results (e.g., wrong class labels) given specially-crafted inputs (i.e., adversarial examples). NMT models are no exception; they can be fooled by adversarial examples [22] or natural noise (e.g., typos in input sentences) [4]. These inferior results (i.e., sub-optimal or incorrect translations), can lead to misunderstanding, embarrassment, financial loss, medical misdiagnoses, threats to personal safety, or political conflicts [17, 57, 64, 65, 80]. Thus, assuring the robustness of machine translation software is an important endeavor.

Yet testing machine translation software is extremely challenging. First, different from traditional software whose logic is encoded in source code, machine translation software is based on complex neural networks with millions of parameters. Therefore, testing techniques for traditional software, which are mostly code-based, are ineffective. Second, the line of recent research on testing artificial intelligence (AI) software [2, 29, 36, 37, 46, 62, 68] focuses on tasks with much simpler output formats—for example, testing image classifiers, which output class labels given an image. However, as one of the most difficult natural language processing (NLP) tasks, the output of machine translation systems (i.e., translated sentences) is significantly more complex. Because they are not structured to handle such complex outputs, when applied to NMT models, typical AI testing approaches almost solely find "illegal" inputs, such as sentences with syntax errors or obvious misspellings that are unlikely given as input. Yet these errors are not the problematic ones in practice; as reported by WeChat, a messenger app with over one billion monthly active users, its embedded NMT model can return inferior results even when the input sentences are syntactically correct [96]. Due to the difficulty of building an effective, automated approach to evaluate the correctness of translation, current approaches for testing machine translation software have many shortcomings.

Approaches that try to address these aforementioned problems still have their own deficiencies—namely, the inability to detect grammatical errors and the lack of real-world test cases. Current testing procedures for machine translation software typically involve three steps [96]: (1) collecting bilingual sentence pairs[1] and

[1] By a sentence pair, we refer to a source sentence and its corresponding target sentence.

Source sentence	Google Translate result	Target sentence meaning	
I live on campus with smart people.	我和聪明的人住在校园里。	I live on campus with smart people.	✓
I live on campus with cute people.	我和可爱的人住在校园里。	I live on campus with cute people.	✓
I live on campus with tall people.	我住在校园里，身材高大。	I live on campus, I am tall.	✗

Figure 1: Examples of similar source sentences and Google Translate results.

splitting them into training, validation, and testing data; (2) calculating translation quality scores (e.g., BLEU [67] and ROUGE [48]) of the trained NMT model on the testing data; and (3) comparing the scores with predefined thresholds to determine whether the test cases pass. However, testing based on a threshold score like BLEU, which is a measurement of the overlap between n-grams of the target and reference, can easily overlook grammatical errors. Additionally, the calculation of translation quality scores (e.g., BLEU) requires bilingual sentence pairs as input, which need to be manually constructed beforehand. To test with real-world user input outside of the training set, extensive manual effort is needed for ground-truth translations. Thus, an effective and efficient testing methodology that can automatically detect errors[2] in machine translation software is in high demand.

To address the above problems, we introduce structure-invariant testing (SIT), a novel, widely-applicable methodology for validating machine translation software. The key insight is that similar source sentences—e.g. sentences that differ by a single word—typically have translation results of similar sentence structures. For example, Fig. 1 shows three similar source sentences in English and their target sentences in Chinese. The first two translations are correct, while the third is not. We can observe that the structure of the third sentence in Chinese significantly differs from those of the other two. For each source sentence, SIT (1) generates a list of its similar sentences by modifying a single word in the source sentence via NLP techniques (i.e., BERT [19]); (2) feeds all the sentences to the software under test to obtain their translations; (3) uses specialized data structures (i.e., constituency parse tree and dependency parse tree) to represent the syntax structure of each of the translated sentences; and (4) compares the structures of the translated sentences. If a large difference exists between the structures of the translated original and any of the translated modified sentences, we report the modified sentence pair along with the original sentence pair as potential errors.

We apply SIT to test Google Translate and Bing Microsoft Translator with 200 source sentences crawled from the Web as input. SIT successfully found 64 buggy issues (defined in Section 3) in Google Translate and 70 buggy issues in Bing Microsoft Translator with high accuracy (i.e., 69.5% and 70% top-1 accuracy respectively). The reported errors[3] are diverse, including under-translation, over-translation, incorrect modification, word/phrase mistranslation, and unclear logic, none of which could be detected by the widely-used metrics BLEU and ROUGE. Examples of different translation errors are illustrated in Fig. 2. The source code and datasets are also released for reuse. Note that our results were *w.r.t.* the snapshots of Google Translate and Bing Microsoft Translator when we performed our testing. After releasing our results dataset in July 2019, we notice that some of the reported translation errors have recently been addressed.

This paper makes the following main contributions:

- It introduces structure-invariant testing (SIT), a novel, widely applicable methodology for validating machine translation software;
- It describes a practical implementation of SIT by adapting BERT [19] to generate similar sentences and leveraging syntax parsers to represent sentence structures;
- It presents the evaluation of SIT using only 200 source sentences crawled from the Web to successfully find 64 buggy issues in Google Translate and 70 buggy issues in Bing Microsoft Translator with high accuracy; and
- It discusses the diverse error categories found by SIT, of which none could be found by state-of-the-art metrics.

2 A REAL-WORLD EXAMPLE

Tom planned to take his son David, who is 14 years old, to the Zurich Zoo. Before their zoo visit, he checked the zoo's website[4] about purchasing tickets and saw the following German sentence:

> Kinder bis 15 Jahre erhalten an ihrem Geburtstag gegen Vorweisen eines gültigen Ausweises den Zooeintritt geschenkt.

Tom is from the United States, and he does not understand German. To figure out its meaning, Tom used Google Translate, a popular translation service powered by NMT models [86]. Google Translate returned the following English sentence:

> Children up to the age of 15 are given free admission to the zoo on presentation of a valid ID.

However, David was denied free entry by the zoo staff even with a valid ID. They found out that they had misunderstood the zoo's regulation because of the incorrect translation returned by Google Translate. The correct translation should be:

> Children up to the age of 15 are given free admission to the zoo *on their birthday* on presentation of a valid ID.

This is a real translation error that led to a confusing, unpleasant experience. Translation errors could also cause extremely serious consequences [17, 57, 65, 80]. For example, a Palestinian man was arrested by Israeli police for a post saying "good morning," which Facebook's machine translation service erroneously translated as

[2]By a translation error, we refer to mistranslation of some parts of a source sentence. The translated sentence (i.e., target sentence) containing translation error(s) is regarded as a buggy sentence. We use "error in the target sentence" and "error in the sentence pair" interchangeably in this paper.

[3]https://github.com/PinjiaHe/StructureInvariantTesting

[4]https://www.zoo.ch/de/zoobesuch/tickets-preise

Error type	Source sentence	Target sentence	Target sentence meaning
Under-translation	It is believed in the field that Amazon employs more PhD economists than any other tech company.	亚马逊聘请的博士经济学家比其他任何科技公司都要多。(by Google)	Amazon employs more PhD economists than any other tech company.
Over-translation	Entering talks, Brazil hoped to see itself elevated to major non NATO ally status by the Trump administration, a big step that would help it purchase military equipment.	进入谈判，巴西希望看到自己被特朗普政府提升为主要的非北约盟国地位，这是一个帮助其购买军事装备的一大步。(by Bing)	Entering talks, Brazil hoped to see itself elevated to major non NATO ally status by the Trump administration, one a big step that would help it purchase military equipment.
Incorrect modification	But even so, they remain prisoners of privilege.	但即便如此，他们仍然是囚犯的特权 (by Google)	But even so, they remain prisoners' priviliege.
Word/phrase mistranslation	I am very willing to share my point of view.	我非常愿意同意我的观点。(by Bing)	I am very willing to agree with my point of view.
Unclear logic	I had a joke to tell and I wanted to finish it, Draper says.	德雷珀说，我开玩笑说，我想完成它。(by Google)	I joked that I want to finish it, Draper says.

Figure 2: Examples of translation errors (English-to-Chinese) detected by SIT.

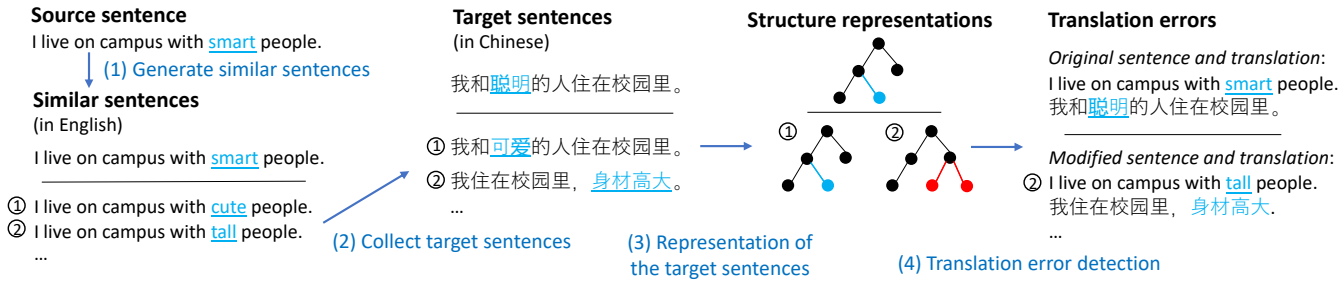

Figure 3: Overview of SIT.

"attack them" in Hebrew and "hurt them" in English [17, 65]. This demonstrates both the widespread reliance on machine translation software and the potential negative effects when it fails. To enhance the reliability of machine translation software, this paper introduces a general validation approach called structure-invariant testing, which automatically and accurately detects translation errors without oracles.

3 APPROACH AND IMPLEMENTATION

This section introduces structure-invariant testing (SIT) and describes our implementation. The input of SIT is a list of unlabeled, monolingual sentences, while its output is a list of suspicious *issues*. For each original sentence, SIT reports either 0 (i.e., no buggy sentence is found) or 1 *issue* (i.e., at least 1 buggy sentence is found). Each *issue* contains: (1) the original source sentence and its translation; and (2) top-k farthest[5] generated source sentences and their translations. The original sentence pair is reported for the following reasons: (1) seeing how the original sentence was modified may help the user understand why the translation system made a mistake; (2) the error may actually lie in the translation of the original sentence.

Fig. 3 illustrates the overview of SIT. In this figure, we use one source sentence as input for simplicity and clarity. The key insight of SIT is that similar source sentences often have target sentences of similar syntactic structure. Derived from this insight, SIT carries out the following four steps:

(1) *Generating similar sentences.* For each source sentence, we generate a list of its similar sentences by modifying a single word in the sentence.
(2) *Collecting target sentences.* We feed the original and the generated similar sentences to the machine translation system under test and collect their target sentences.
(3) *Representing target sentence structures.* All the target sentences are encoded as data structures specialized for natural language processing.
(4) *Detecting translation errors.* The structures of the translated modified sentences are compared to the structure of the

[5]the distance metric here is between the structures of the original sentence translation and the modified sentence translations

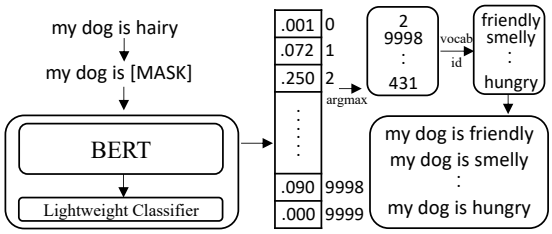

Figure 4: Similar sentence generation process.

translated original sentence. If there is a large difference between the structures, SIT reports a potential error.

3.1 Generating Similar Sentences

In order to test for structural invariance, we must compare two sentences that have the same syntactic structure but differ in at least one token. We have found that, given an input sentence, changing one word in the sentence at a time under certain constraints effectively produces a set of structurally identical and semantically similar sentences.

Explicitly, the approach we take modifies a single token in an input sentence, replacing it with another token of the same part of speech (POS),[6] to produce an alternate sentence. For example, we will mask "hairy" in the source sentence in Fig. 4 and replace it with the top-k most similar tokens to generate k similar sentences. We do this for every candidate token in the sentence; for the sake of simplicity and to avoid grammatically strange or incorrect sentences, we only use nouns and adjectives as candidate tokens.

Now we discuss the problem of selecting replacement tokens. Perhaps the simplest algorithm for selecting a set of replacement tokens would involve using word embeddings [60]. One could choose words that have high vector similarity with and identical POS tags to a given token in the original sentence as replacements in the modified sentences. However, since word embeddings have the same value regardless of context, this approach often produces sentences that would not occur in common language. For example, the word "fork" might have high vector similarity with and the same POS tag as the word "plate." However, while the sentence "He came to a fork in the road" makes sense, the sentence "He came to a plate in the road" does not.

Rather, we want a model that considers the surrounding words and comes up with a set of replacements that, when inserted, create realistic sentences. A model that does just this is the masked language model (MLM) [59], inspired by the Cloze task [78]. The input to an MLM is a piece of text with a single token masked (i.e., deleted from the sentence and replaced with a special indicator token). The job of the model is then to predict the token in that position given the context. This method forces the model to learn the dependencies between different tokens. Since there are a number of different contexts a single word can fit in, this model, in a sense, allows for a single token to have multiple representations. We therefore get a set of replacement tokens that are context dependent. While the predicted tokens are not guaranteed to have the same meaning as the original token, if the MLM is well trained, it is highly likely that the sentence with the new, predicted token is both syntactically correct and meaningful.

An example of the sentence generation process is demonstrated in Fig. 4. For our implementation, we use BERT [19], which is a state-of-the-art language representation model recently proposed by Google. The out-of-box BERT model provides pre-trained language representations that can be fine-tuned by adding an additional lightweight softmax classification layer to create models for a wide range of language-related tasks, such as masked language modelling. BERT was trained on a huge amount of data—a concatenation of BooksCorpus (800M words) and English Wikipedia (2,500M words)—with the masked language task being one of two main tasks used for training. Thus, we believe that BERT fits this aspect of our approach well.

3.2 Collecting Target Sentences

Once we have generated a list of similar sentences from our original sentence, the next step is to input all the source sentences to the machine translation software under test and collect the corresponding translation results (i.e., target sentences). We subsequently analyze the results to find errors. We use Google's and Bing's machine translation systems as test systems for our experiment. To obtain translation results, we invoke the APIs provided by Google Translate and Bing Microsoft Translator, which return identical results as their Web interfaces [5, 30].

3.3 Representations of the Target Sentences

Next we must model the target sentences obtained from the translation system under test as this allows us to compare structures to detect errors. Choosing the structure with which to represent our sentences will affect our ability to perform meaningful comparisons. We ultimately want a representation that precisely models the structure of a sentence while offering fast comparison between two values.

The simplest and fastest approach is to compare sentences in their raw form: as strings. Indeed, we test this method and performance is reasonable. However, there are many scenarios in which this method falls short. For example, the prepositional phrase "on Friday" in the sentence "On Friday, we went to the movies" can also be placed on the end of the sentence as follows: "We went to the movies on Friday." The sentences are interchangeable but a metric such as character edit distance would indicate a large difference between the strings. Syntax parsing overcomes the above issue. With a syntax parser, we can model the syntactic structure of a string and the relationship between words or groups of words. For example, if parsing is done correctly, our two sample sentences above should have identical representations in terms of relation values and parse structure.

3.3.1 Raw Target Sentence. For this method, we leave our target sentence in its original format, i.e., as a string. In most cases, we may expect that editing a single token in a sentence in one language would lead to the change of a single token in the translated sentence. Given the syntactic role of the replacement token is the same, this would ideally happen in all machine translation systems. However, this is not always the case in practice as prepositional phrases, modifiers, and other constituents can often be

[6] https://en.wikipedia.org/wiki/Part_of_speech

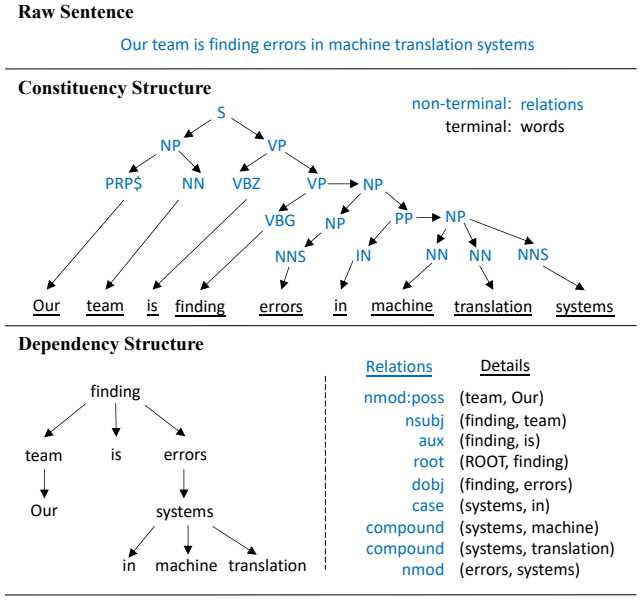

Figure 5: Representing sentence structures; both dependency & constituency relations can be displayed as trees.

placed in different locations by the translation system and produce a semantically-equivalent, grammatically correct sentence. Nonetheless, this method serves as a good baseline.

3.3.2 Constituency Parse Tree. Constituency parsing is one method for deriving the syntactic structure of a string. It generates a set of constituency relations, which show how a word or group of words form different units within a sentence. This set of relations is particularly useful for SIT because it will reflect changes to the type of phrases in a sentence. For example, while a prepositional phrase can be placed in multiple locations to produce a sentence with the same meaning, the set of constituency relations will remain unchanged. Constituency relations can be visualized as a tree, as shown in Fig. 5. A constituency parse tree is an ordered, rooted tree where non-terminal nodes are the constituent relations and terminal nodes are the words. Formally, in constituency parsing, a sentence is broken down into its constituent parts according to the phrase structure rules [14] outlined by a given context-free grammar. For our experiments, we use the shift-reduce constituency parser by Zhu *et al.* [99] and implemented in Stanford's CoreNLP library [31]. It can parse about 50 sentences per second.

3.3.3 Dependency Parse Tree. Dependency parsing likewise derives the syntactic structure of a string. However, the set of relations produced describe the direct relationships between words rather than how words constitute a sentence. This set of relations gives us different insights about structure and is intuitively useful for SIT because it will reflect changes between how words interact. Much progress has been made over the past 15 years on dependency parsing. Speed and accuracy increased dramatically with the introduction of neural network based parsers [11]. As with shift-reduce constituency parsers, neural network based dependency parsers use a stack-like system where transitions are chosen using a classifier. The classifier in this case is a neural network, likewise trained on annotated tree banks. For our implementation, we use the most recent neural network based parsers made available by Stanford CoreNLP, which can parse about 100 sentences per second. We use the Universal Dependencies as our annotation scheme, which has evolved based off the Stanford Dependencies [18].

3.4 Translation Error Detection via Structure Comparison

Finally, in order to find translation errors, we search for structural variation by comparing sentence representations. Whether sentences are modelled as raw strings, word embeddings, or parse trees, there are a number of different metrics for calculating the distance between two values. These metrics tend to be quite domain specific and might have low correlation with each other, making the choice of metric incredibly important. For example, a metric such as Word Mover's Distance [41] would give us a distance of 0 between the two sentences "He went to the store" and "Store he the went to" while character edit distance would give a distance of 14. We explore several different metrics for evaluating the distance between sentences: character (Levenshtein) edit distance, constituency set difference, and dependency set difference.

3.4.1 Levenshtein Distance between Raw Sentences. The Levenshtein distance [44], sometimes more generally referred to as the "edit distance," compares two strings and determines how closely they match each other by calculating the minimum number of character edits (deletions, insertions, and substitutions) needed to transform one string into the other. While the method may not demonstrate syntactic similarity between sentences well, it exploits the expectation that editing a single token in a sentence in one language will often lead to the change of only a single token in the translated sentence. Therefore, the Levenshtein distance serves as a good baseline metric.

3.4.2 Relation Distance between Constituency Parse Trees. To evaluate the distance between two sets of constituency relations, we calculate the distance between two lists of constituency grammars as the sum of absolute difference in the count of each phrasal type, which gives us a basic understanding of how a sentence has changed after modification. The motivation behind this heuristic is that the constituents of a sentence should stay the same between two sentences where only a single token of the same part of speech differs. In a robust machine translation system, this should be reflected in the target sentences as well.

3.4.3 Relation Distance between Dependency Parse Trees. Similarly, for calculating the distance between two lists of dependencies, we sum the absolute difference in the number of each type of dependency relations. Again, the motivation is that the relationships between words will ideally remain unchanged when a single token is replaced. Therefore, a change in the set is reasonable indication that structural invariance has been violated and presumably there is a translation error.

3.4.4 Distance Thresholding. Using one of the above metrics, we calculate the distance between the original target sentence and the generated target sentences. We must then decide whether a

modified target sentence is far enough from the its corresponding original target sentence to indicate the presence of a translation error. To do this, we first filter based on a distance threshold, only keeping sentences that are farther from the original sentence than the chosen threshold. Then, for a given original target sentence, we report the top-k (k also being a chosen parameter) farthest modified target sentences. We leave the distance threshold as a manual parameter since the user may prioritize minimizing false positive reports or minimizing false negative reports depending on their goal. In Section 4.6, we show tradeoffs for different threshold values. For each original sentence, an issue will be reported if at least one translated generated sentence is considered buggy.

4 EVALUATION

In this section, we evaluate our approach by applying it to Google Translate and Bing Microsoft Translator with real-world unlabeled sentences crawled from the Web. Our main research questions are:

- RQ1: How effective is the approach at finding buggy translations in machine translation software?
- RQ2: What kinds of translation errors can our approach find?
- RQ3: How efficient is the approach?
- RQ4: How do we select the distance threshold in practice?

4.1 Experimental Setup

To verify the results of SIT, we manually inspect each issue reported and collectively decide: (1) whether the issue contains buggy sentences; and (2) if yes, what kind of translation errors it contains. All experiments are run on a Linux workstation with 6 Core Intel Core i7-8700 3.2GHz Processor, 16GB DDR4 2666MHz Memory, and GeForce GTX 1070 GPU. The Linux workstation is running 64-bit Ubuntu 18.04.02 with Linux kernel 4.25.0.

4.2 Dataset

Typically, to test a machine translation system, developers can adopt SIT with any source sentence as input. Thus, to evaluate the effectiveness of our approach, we collect real-world source sentences from the Web. Specifically, input sentences are extracted from CNN[7] (Cable News Network) articles in two categories: politics and business. The datasets are collected from two categories of articles because we intend to evaluate whether SIT consistently performs well on sentences of different semantic context.

For each category, we crawled the 10 latest articles, extracted their main text contents, and split them into a list of sentences. Then, we randomly select 100 sentences from each sentence list as the experimental datasets (200 in total). In this process, sentences that contain more than 35 words are filtered because we intend to demonstrate that machine translation software can return inferior results even for relatively short, simple sentences. The details of the collected datasets are illustrated in Table 1.

4.3 The Effectiveness of SIT

Our approach aims to automatically find translation errors using unlabeled sentences and report them to developers. Thus, the effectiveness of the approach lies in two aspects: (1) how accurate are the reported results; and (2) how many buggy sentences can SIT find? In this section, we evaluate both aspects by applying SIT to test Google Translate and Bing Microsoft Translator using the datasets illustrated in Table 1.

[7] https://edition.cnn.com/

Table 1: Statistics of input sentences for evaluation. Each corpus contains 100 sentences.

Corpus	# of Words/ Sentence	Average # of Words/Sentence	# of Words Total	# of Words Distinct
Politics	4~32	19.2	1,918	933
Business	4~33	19.5	1,949	944

Table 2: Top-k accuracy of SIT.

Google Translate	Top-1 (#buggy issues)	Top-2 (#buggy issues)	Top-3 (#buggy issues)
SIT (Raw)	55.0% (55)	63.0% (63)	66.0% (66)
SIT (Constituency)	61.3% (62)	66.3% (67)	68.3% (69)
SIT (Dependency)	**69.5% (64)**	**71.7% (66)**	**73.9% (68)**
Bing Microsoft Translator	**Top-1 (#buggy issues)**	**Top-2 (#buggy issues)**	**Top-3 (#buggy issues)**
SIT (Raw)	58.8% (60)	69.6% (71)	71.5% (73)
SIT (Constituency)	67.0% (67)	**71.0% (71)**	74.0% (74)
SIT (Dependency)	**70.0% (70)**	**71.0% (71)**	**78.0% (78)**

4.3.1 Evaluation Metric. The output of SIT is a list of *issues*, each containing (1) an original source sentence and its translation; (2) the top-k reported generated sentences and their translations (i.e. the k farthest translations from the source sentence translation). Here we define top-k accuracy as the percentage of reported issues where at least one of the top-k reported sentences or the original sentence contains an error. We use this as our accuracy metric for SIT. Explicitly, if there is a buggy sentence in the top-k generated sentences of issue i, we consider the issue to be accurate and set $buggy(i,k)$ to true; else we set $buggy(i,k)$ to false. If the original sentence is buggy and was reported as an issue, then we also set $buggy(i,k)$ to true. Given a list of issues I, its top-k accuracy is calculated as:

$$\text{Accuracy}_k = \frac{\sum_{i \in I} \mathbb{1}\{buggy(i,k)\}}{|I|}, \tag{1}$$

where $|I|$ is the number of the issues returned by SIT.

4.3.2 Results. **Top-k accuracy.** The results are summarized in Table 2. SIT (Raw), SIT (Constituency), and SIT (Dependency) are SIT implementations with raw sentence, constituency structure, and dependency structure as sentence structure representation, respectively. Each item in the table presents the top-k accuracy along with the number of buggy issues found. In subsequent discussions, for brevity, we refer SIT (Constituency) and SIT (Dependency) as SIT (Con) and SIT (Dep), respectively.

We observe that SIT (Con) and SIT (Dep) consistently perform better than SIT (Raw), which demonstrates the importance of the structure representation of sentences. The metric used in SIT (Raw),

Table 3: Number of unique errors. Top-k unique errors by SIT are errors only in generated sentences output by SIT (Dep).

	Original sentences	#Top-1 unique errors by SIT	#Top-2 unique errors by SIT	#Top-3 unique errors by SIT
Google	55	45	64	79
Bing	60	32	43	66

Table 4: Number of sentences that have specific errors in each category SIT (Dep).

Google \ Bing	Under translation	Over translation	Incorrect modification	Word/phrase mistranslation	Unclear logic
Top-1	35 \ 17	9 \ 8	4 \ 2	44 \ 54	27 \ 31
Top-2	48 \ 23	12 \ 15	6 \ 3	59 \ 60	44 \ 41
Top-3	61 \ 35	15 \ 21	10 \ 4	75 \ 93	53 \ 59

which is based only on the characters in the sentences, is brittle and subject to over and under report errors. For example, SIT (raw) may report sentences that are different in word level but similar in sentence structure, leading to false positives. SIT (Con) and SIT (Dep) achieve comparable performance in terms of both top-k accuracy and the number of reported buggy issues. In particular, when testing Bing Microsoft Translator, SIT (Dep) reports 100 suspicious issues. Among these issues, 70 of them contain translation errors in the first reported sentence or the original sentence, achieving 70% top-1 accuracy. SIT (Dep) has the best performance on Top-1 accuracy for both Google Translate and Bing Microsoft Translator. It successfully finds 64 and 70 buggy issues with 69.5% and 71% top-1 accuracy, respectively. SIT (Dep) also achieves the highest top-3 accuracy (73.9% and 78%). Note that source sentences in the same issue only differ by one word. Thus, inspecting top-3 sentences will not cause more effort compared with inspecting top-1 sentences.

In addition, we study whether SIT can trigger new errors in the generated sentences. As illustrated in Table 3, in the reported issues, 55 and 60 unique errors are found in the translation of original sentences by Google Translate and Bing Microsoft Translator respectively. Besides these errors, SIT finds 79 and 66 extra unique errors that are revealed only in the generated sentence pairs but not in the original. Thus, given its high top-k accuracy and lots of extra unique errors reported, we believe SIT is very useful in practice.

We did not compared SIT's accuracy with [96] and [97] because of the following reasons. SIT targets general mistranslation errors, while [96] focuses on under-/over-translations. Thus, we did not empirically compare with it. In terms of error type and quantity, [96] can only find some under-/over-translation errors in original sentence translations, while SIT finds general errors in translations of both original sentences and their derived similar sentences. [97] requires input sentences with specialized structures and thus it cannot detect any errors using our datasets.

4.4 Translation Error Reported by SIT

SIT is capable of finding translation errors of diverse kinds. In our experiments with Google Translate and Bing Microsoft Translator, we mainly find 5 kinds of translation errors: under-translation, over-translation, incorrect modification, word/phrase mistranslation, and unclear logic. The error types are derived from error classification methods for machine translation. Each of the five is a subset of lexical, syntactic, or semantic errors [34]. We rename them in a more intuitive manner to aid the readers. To provide a glimpse of the diversity of the uncovered errors, this section highlights examples for all the 5 kinds of errors. Table 4 presents the statistics of the translation errors SIT found. Under-translation, word/phrase mistranslation, and unclear logic account for most of the translation errors found by SIT.

4.4.1 Under-Translation. If some words are mistakenly untranslated (i.e. do not appear in the translation), it is an under-translation error. Fig. 6 presents a sentence pair that contains under-translation error. In this example, "to Congress" is mistakenly untranslated, which leads to target sentences of different semantic meaning. Specifically, "lying to Congress" is illegal while "lying" is just an inappropriate behavior. Likewise, the real-world example introduced in Section 2 is caused by an under-translation error.

Source	After pleading guilty in the Manhattan probe, Cohen also later pleaded guilty to lying **to Congress** in a case brought by Mueller's website.
Target	在曼哈顿调查中认罪后，科恩后来还对穆勒网站提起的一起案件中的撒谎罪供认不讳。(by Bing)
Target meaning	After pleading guilty in the Manhattan probe, Cohen also later pleaded guilty to lying in a case brought by Mueller's website.

Figure 6: Example of under-translation errors detected.

4.4.2 Over-Translation. If some words are unnecessarily translated multiple times or some words in the target sentence are not translated from any words in the source sentence, it is an over-translation error. In Fig. 7, "thought" in the target sentence is not translated from any words in the source sentence, so it is an over-translation error. Interestingly, we found that an over-translation error often happens along with some other kinds of errors. The example also contains an under-translation error because "were right" in the source sentence is mistakenly untranslated. In the second example in Fig. 2, the word "a" is unnecessarily translated twice, which makes it an over-translation error.

Source	The investigators were right that the airplane itself was safe.
Target	调查人员认为飞机本身是安全的。(by Google)
Target meaning	The investigators **thought** that the airplane itself was safe.

Figure 7: Example of over-translation errors detected.

4.4.3 Incorrect Modification. If some modifiers modify the wrong element in the sentence, it is an incorrect modification error. In Fig. 8, the modifier "new" modifies "auto manufacturing" in the source sentence. However, Google Translate thinks that "new" should modify "hub." In Fig. 2, the third example also shows an interesting incorrect modification error. In this example ("prisoners of privilege"), "privilege" modifies "prisoners" in the source sentence, while Google Translate thinks "prisoners" should modify "privilege." We think that in the training data of the NMT model, there are some phrases with the similar pattern: "*A* of *B*," where *A* modifies B, which leads to an incorrect modification error in this scenario. Interestingly, the original source sentence that triggers this error is "But even so, they remain *bastions of privilege*." In the original sentence, "bastions" modifies "privilege," which fits the supposed archetype. As we might expect, this sentence is correctly translated by Google Translate.

Source	The South has emerged as a hub of new auto manufacturing by foreign makers thanks to lower manufacturing costs and less powerful businesses.
Target	由于制造成本降低和业务不那么强大，南方已成为外国制造商新的汽车制造中心。(by Google)
Target meaning	The South has emerged as a new hub of auto manufacturing by foreign makers thanks to the reducing manufacturing costs and less powerful businesses.

Figure 8: Example of incorrect modification errors detected.

4.4.4 Word/phrase Mistranslation. If some tokens or phrases are incorrectly translated in the target sentence, it is a word/phrase mistranslation error. Fig. 9 presents two main sub-categories of this kind of error: (1) ambiguity of polysemy and (2) wrong translation.

Ambiguity of polysemy. Each token/phrase may have multiple correct translations. For example, admit means "allow somebody to join an organization" or "agree with something unwillingly." However, usually in a specific semantic context (e.g., a sentence), a token/phrase only has one correct translation. Modern translation software does not perform well on polysemy. In the first example in Fig. 9, Google Translate thinks the "admit" in the source sentence refers to "agree with something unwillingly," leading to a token/phrase mistranslation error.

Wrong translation. A token/phrase could also be incorrectly translated to another meaning that seems semantically unrelated. For example, in the second example in Fig. 9, Bing Microsoft Translator thinks "South" refers to "South Korea," leading to a word/phrase mistranslation error.

4.4.5 Unclear Logic. If all the tokens/phrases are correctly translated but the sentence logic is incorrect, it is an unclear logic error. In Fig. 10, Google Translate correctly translates "serving in the elected office" and "country." However, Google Translate generates "serving in the elected office as a country" instead of "serving the country in elected office" because Google Translate does not understand the logical relation between them. Unclear logic errors exist widely in translations given by NMT models, which is to some extent a sign of whether a model truly understands certain semantic meanings.

Source	The most elite public universities admit a considerably larger percentage of students from lower income backgrounds than do the elite private schools.
Target	最精英的公立大学承认，与精英私立学校相比，低收入学生的比例要高得多。(by Google)
Target meaning	The most elite public universities agree unwillingly that considerably larger percentage of students from lower income backgrounds than do the elite private schools.
Source	The South has emerged as a hub of new auto manufacturing by foreign makers thanks to lower manufacturing costs and less powerful unions.
Target	由于制造成本较低，工会实力较弱，韩国已成为外国制造商新汽车制造业的枢纽。(by Bing)
Target meaning	The South Korea has emerged as a hub of new auto manufacturing by foreign makers thanks to lower manufacturing costs and less powerful unions.

Figure 9: Examples of word/phrase mistranslation errors detected.

Source	And attacking a dead man who spent five years as a prisoner of war and another three decades serving the country in elected office, is simply wrong.
Target	并且攻击一名死去的人，他在战争中担任战争囚犯五年，另外三十年担任民选职务的国家，这是完全错误的。(by Google)
Target meaning	And attacking a dead man who spent five years as a prisoner of war and another three decades serving in elected office as a country, is simply wrong.

Figure 10: Example of unclear logic errors detected.

4.4.6 Sentences with Multiple Translation Errors. A certain percentage of reported sentence pairs contain multiple translation errors. Fig. 11 presents a sentence pair that contains three kinds of errors. Specifically, "covering" means "reporting news" in the source sentence. However, it is translated to "holding," leading to a word/phrase mistranslation error. Additionally, "church" in the target sentence is not the translation of any words from the source sentence, so it is an over-translation error. Bing Microsoft translator also wrongly thinks the subject is "attending a funeral train." But the source sentence actually means the subject is "covering a funeral train," so it is an unclear logic error.

Errors	word/phrase logic *over*
Source	Covering a memorial service in the nation's capital and then traveling to Texas for another service as well as a funeral train was an honor, he says.
Target	他说，在美国首都举行追悼会，然后前往德州参加另一次礼拜仪式以及葬礼列车，是一种荣誉。(by Bing)
Target meaning	Holding a memorial service in the nation's capital and then traveling to Texas for attending another *church* service and a funeral train was an honor, he says.

Figure 11: Example of sentence with multiple translation errors detected.

4.5 The Running Time of SIT

In this section, we evaluate the running time of SIT on the two datasets. We apply SIT with 3 different sentence structure representations to test Google Translate and Bing Microsoft Translator. We run each experiment setting 10 times and report their average as the results. The overall running time of SIT is illustrated in Table 5, and the running time of each step of SIT on Google Translate is presented in Fig. 12 (Bing's result is similar). We can observe that SIT using raw sentences as structure representation is the fastest. This is because SIT (Raw) does not require any structure representation generation time. SIT using a dependency parser achieves comparable running time to SIT (Raw). In particular, SIT (Dep) uses 19 seconds to parse 2000+ sentences (as opposed to 0 seconds by SIT (Raw)), which we think is efficient and reasonable.

Table 5: Average running time of SIT on Politics and Business datasets.

Google \ Bing	Running time (sec)	Translation time (sec)	#Sentence translated	Time of other SIT steps (sec)
SIT (Raw)	1,469 \ 922	1,417 \ 870	2,012	52 \ 52
SIT (Constituency)	1,524 \ 981	1,417 \ 870	2,012	107 \ 110
SIT (Dependency)	1,488 \ 945	1,417 \ 870	2,012	71 \ 75

In these experiments, we ran the translation step once per translation system and reused the translation results in all experiment settings since the other settings had no impact on translation time. Thus, in Table 5, the *Translation time* values are the same for different SIT implementations. We can observe that SIT spends most of the time collecting translation results. In this step, for each sentence, we invoked the APIs provided by Google and Bing to collect the translated sentence. In practice, if users want to test their own machine translation software with SIT, the running time of this step will be much less. As indicated in a recent study [92], current NMT model can translate around 20 sentences per second using a single NVIDIA GeForce GTX 1080 GPU. With more powerful computing resource (e.g, TPU [86]), modern NMT models can achieve the speed of hundreds of sentences translation per second, which would be about 2 magnitudes faster than in our experiments.

The other steps of SIT are quite efficient, as indicated in Table 5 and Fig. 12. Both SIT (Raw) and SIT (Dep) took around 1 min and SIT (Con) took around 2 mins. Compared with SIT (Dep), SIT (Con) is slower because models for constituency parsing are slower than those for dependency parsing. We conclude that as a tool working in an offline manner, SIT is efficient in practice for testing machine translation software.

4.6 The Impact of Distance Threshold

SIT reports the top-k sentence pairs in an issue if the distance between the translated generated sentence and the original target sentence is larger than a distance threshold. Thus, this distance threshold controls (1) the number of buggy issues reported and (2) the top-k accuracy of SIT. Intuitively, if we lower the threshold, more buggy issues will be reported, while the accuracy will decrease. Fig. 13 demonstrates the impact of the distance threshold on these two factors. In this figure, SIT (Dep) was applied to test

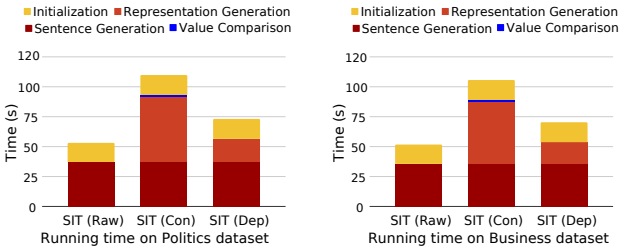

Figure 12: Running time details of SIT (excluding translation time) in testing Google Translate.

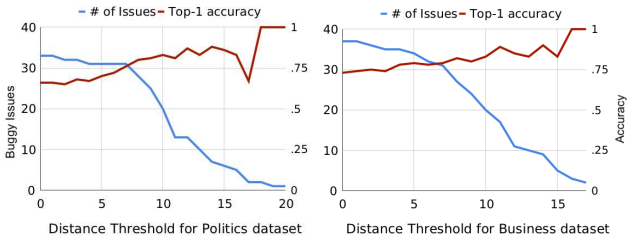

Figure 13: Impact of distance threshold when testing Bing Microsoft Translator.

the Bing Microsoft Translator on our Politics and Business datasets with different distance thresholds. We can observe that both the number of buggy issues and top-1 accuracy remain stable when the threshold is either small or large while the values fluctuate in the middle. The impact of changing the distance threshold is similar when testing Google Translate.

Based on these results, we present some guidance on using SIT in practice. First, if we intend to uncover as many translation errors as possible, we should use a small distance threshold. A small threshold (e.g., 4 for dependency sets) works well on all our experiment settings. In particular, with a small threshold, SIT reports the most issues with decent accuracy (e.g., 70% top-1 accuracy). We adopt this strategy in our accuracy experiments in Section 4.3.2. Developers could use SIT with small distance threshold when they want to intensively test software before a release. Second, if we intend to make SIT as accurate as possible, we could use a large threshold (e.g., 15). With a large threshold, SIT reports fewer issues with very high accuracy (e.g., 86% top-1 accuracy). Given that the number of source sentences are unlimited on the Web, we could keep running SIT with a large distance threshold and periodically report issues. Thus, we think SIT is effective and easy to use in practice.

4.7 Fine-tuning with Errors Reported by SIT

In this section, we study whether the reported buggy sentences can act as a fine-tuning set to improve the robustness of NMT models. Fine-tuning is a common practice in NMT, where training data and target data can often occupy different domains [15, 74]. Specifically, we train an encoder-decoder model with global attention [51]—a standard architecture for NMT models—on a subset of the CWMT

corpus with 2M bilingual sentence pairs [16]. The encoder and decoder are unidirectional single-layer LSTMs. We train the model using the Adam optimizer [40], calculating the BLEU [67] score on a held out validation set after each epoch. We use the model with parameters from the epoch with the best validation BLEU score. Note that we did not use Google or Bing's translation models here because they are not open-source; however, the encoder-decoder model with attention is a very representative NMT model.

To test the NMT model, SIT is run on 40 English sentences, which are selected from the validation set of WMT'17 [85] by removing long sentences (i.e., longer than 12 words) and ensuring that all words are in the NMT model's vocabulary. Note that since the model was not trained or validated on data from this domain, we simulated the practical scenario where real-world inputs differ from model training data. Based on these inputs, SIT successfully finds 105 buggy sentences. We label them with correct translations and fine-tune the NMT model on these 105 sentences for 15 epochs with a decreasing learning rate. After this fine-tuning, all the 105 sentences can be correctly translated. Meanwhile, the BLEU score on the original validation set used during training increases by 0.13, which, to some degree, shows that the translation of other sentences has also been improved. This demonstrates the ability to fix errors reported by SIT in an efficient and easy manner. SIT's utility on building robust machine translation software will be further elaborated in Section 5.2.

5 DISCUSSIONS

5.1 False Positives

While SIT can accurately detect translation errors, its precision can be further improved. In particular, the false positives of SIT come from three main sources. First, the generated sentences may have strange semantic meanings, leading to changes in the target sentence structure. For example, based on the phrase "on the way," the current implementation of SIT could generate the sentence "on the fact," which naturally has a very different translation in Chinese. Using BERT, which at the time of our experiments provided the state-of-the-art masked language model, helped alleviate this issue. Second, although the existing syntax parsers are highly accurate, they may produce wrong constituency or dependency structures, which can lead to erroneous reported errors. Third, a source sentence could have multiple correct translations of different sentence structures. For example, target sentence "10 years from now" and "after 10 years" can be used interchangeably while their sentence structures are different. To lower the impact of these factors, SIT returns the top-k suspicious sentence pairs ranked by distance to the original target sentence.

5.2 Building Robust Translation Software

The ultimate goal of testing machine translation, similar to testing traditional software, is to build robust software. Toward this end, SIT's utility is as follows. First, the reported mistranslations typically act as early alarms, and thus developers can hard-code translation mappings in advance, which is the quickest bug fixing solution adopted in industry. Second, the reported sentences could be used as a fine-tuning set, which has been discussed in Section 4.7. Third, developers may find the reported buggy sentence pairs useful for further analysis/debugging since the sentences in each pair only differ by one word. This resembles debugging traditional software via input minimization/localization. Additionally, the structural invariance concept could be utilized as inductive bias to design robust NMT models, similar to how Shen *et al.* [75] introduce bias to standard LSTMs. Compared with traditional software, the debugging and bug fixing process of machine translation software is more difficult because the logic of an NMT model mainly lies in its model structure and parameters. While this is not the main focus of our work, we believe it is an important research direction for future work.

6 RELATED WORK

6.1 Robustness of AI Software

The success of deep learning models has led to the wide adoption of artificial intelligence (AI) software in our daily lives. Despite their high accuracies, deep learning models can generate inferior results, some of which have even lead to fatal accidents [42, 45, 100]. Recently, researchers have designed a variety of approaches to attack deep learning (DL) systems [3, 6, 7, 20, 29, 89, 91]. To protect DL systems against these attacks, excellent research has been conducted to test DL systems [21, 26, 33, 39, 53, 54, 68, 69, 79, 88, 94, 95], assist the debugging process [55], detect adversarial examples online [56, 77, 84, 90], or train networks in a robust way [38, 49, 58, 66]. Compared with these approaches, our paper focuses on machine translation systems, which these works do not explore. In addition, most of these approaches require knowledge of gradients or activation values in the neural network under test (white-box), while our approach does not require any internal details of the model (black-box).

6.2 Robustness of NLP Algorithms

Deep neural networks have boosted the performance of many NLP tasks , such as reading comprehension [9, 10], code analysis [1, 35, 70], and machine translation [32, 83, 86]. However, in recent years, inspired by the work on adversarial examples in the computer vision field, researchers successfully found bugs produced by the neural networks used for various NLP systems [2, 8, 36, 36, 37, 46, 61, 62, 72]. Compared with our approach, these works focus on simpler tasks such as text classification.

Zheng *et al.* [96] introduced two algorithms to detect two specific translation errors: under-translation and over-translation, respectively. Comparatively, our proposed approach is more systematic and not limited to specific errors. Based on the experimental results, we can find the following errors: under-translation, over-translation, incorrect modification, ambiguity of polysemy, and unclear logic. Zhou and Sun [97] proposed a metamorphic testing approach (i.e., MT4MT) for machine translation; they followed a concept similar to structural invariance. However, MT4MT can only be used with simple sentences in a *subject-verb-object* pattern (e.g., "Tom likes Nike"). In particular, they change a person name or a brand name in a sentence and check whether the translation differs by more than one token. Thus, MT4MT cannot report errors from most real-world sentences, such as the data set used in our paper. In addition, MT4MT does not propose general techniques to realize their idea.

Our work introduces an effective realization via nontrivial techniques (e.g., adapting BERT for word substitution and leveraging language parsers for generating sentence structures), and conducts an extensive evaluation.

6.3 Machine Translation

The past few years have witnessed rapid growth for neural machine translation (NMT) architectures [32, 86]. Typically, an NMT model uses an encoder-decoder framework with attention [92]. Under this framework, researchers have designed various advanced neural network architectures, ranging from recurrent neural networks (RNN) [52, 76], convolutional neural networks (CNN) [27, 28], to full attention networks without recurrence or convolution [83]. These existing papers aim at improving the capability of NMT models. Different from them, this paper focuses on the robustness of NMT models. We believe robustness is as important as accuracy for machine translation in practice. Thus, our proposed approach can complement existing machine translation research.

6.4 Metamorphic Testing

Metamorphic testing is a way of generating test cases based on existing ones [12, 13, 73]. The key idea is to detect violations of domain-specific metamorphic relations across outputs from multiple runs of the program with different inputs. Metamorphic testing has been applied for testing various traditional software, such as compilers [43, 47], scientific libraries [93], and database systems [50]. Due to its effectiveness on testing "non-testable" programs, researchers have also used it to test AI software, such as statistical classifiers [63, 87], search engines [98], and autonomous cars [79, 95]. In this paper, we introduce structure-invariant testing, a novel, widely applicable metamorphic testing approach, for machine translation software.

7 CONCLUSION

We have presented structure-invariant testing (SIT), a new, effective approach for testing machine translation software. The distinct benefits of SIT are its simplicity and generality, and thus wide applicability. SIT has been applied to test Google Translate and Bing Microsoft Translators, and successfully found 64 and 70 buggy issues with 69.5% and 70% top-1 accuracy, respectively. Moreover, as a general methodology, SIT can uncover diverse kinds of translation errors that cannot be found by state-of-the-art approaches. We believe that this work is the important, first step toward systematic testing of machine translation software. For future work, we will continue refining the general approach and extend it to other AI software (e.g., figure captioning tools and face recognition systems). We will also launch an extensive effort to help continuously test and improve widely-used translation systems.

ACKNOWLEDGMENTS

We would like to thank the anonymous ICSE reviewers for their valuable feedback on the earlier draft of this paper. In addition, the tool implementation benefited tremendously from Stanford NLP Group's language parsers [31] and Hugging Face's BERT implementation in PyTorch [24].

REFERENCES

[1] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning Distributed Representations of Code. *Proceedings of the ACM on Programming Languages* POPL (2019).
[2] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating Natural Language Adversarial Examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
[3] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*.
[4] Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and Natural Noise Both Break Neural Machine Translation. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.
[5] Bing. [n.d.]. Bing Microsoft Translator. https://www.bing.com/translator
[6] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. 2016. Hidden Voice Commands. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*.
[7] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy*.
[8] Isaac Caswell, Onkur Sen, and Allen Nie. 2015. Exploring adversarial learning on neural network models for text classification. (2015).
[9] Danqi Chen. 2018. *Neural Reading Comprehension and Beyond*. Ph.D. Dissertation. Stanford University.
[10] Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.
[11] Danqi Chen and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
[12] Tsong Y. Chen, Shing C. Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.
[13] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Computing Surveys (CSUR)* 51 (2018). Issue 1.
[14] N. Chomsky. 1957. *Syntactic Structures*. Mouton, The Hague.
[15] Chenhui Chu, Raj Dabre, and Sadao Kurohashi. 2017. An empirical comparison of simple domain adaptation methods for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.
[16] CWMT. [n.d.]. CWMT Datasets. http://nlp.nju.edu.cn/cwmt-wmt/
[17] Gareth Davies. 2017. Palestinian man is arrested by police after posting 'Good morning' in Arabic on Facebook which was wrongly translated as 'attack them'. https://www.dailymail.co.uk/news/article-5005489/Good-morning-Facebook-post-leads-arrest-Palestinian.html
[18] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. [n.d.]. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC)*.
[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).
[20] Tianyu Du, Shouling Ji, Jinfeng Li, Qinchen Gu, Ting Wang, and Raheem Beyah. 2019. SirenAttack: Generating Adversarial Audio for End-to-End Acoustic Systems. *arXiv preprint arXiv:1901.07846* (2019).
[21] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.
[22] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 2018. On Adversarial Examples for Character-Level Neural Machine Translation. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*.
[23] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2018. Robust Physical-World Attacks on Deep Learning Models. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[24] Hugging Face. [n.d.]. Transformers: State-of-the-art Natural Language Processing for TensorFlow 2.0 and PyTorch. https://github.com/huggingface/transformers
[25] Facebook. 2019. How do I translate a post or comment written in another language? https://www.facebook.com/help/509936952489634?helpref=faq_content

[26] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*.

[27] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2017. A Convolutional Encoder Model for Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.

[28] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*.

[29] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

[30] Google. [n.d.]. Google Translate. https://translate.google.com/

[31] Stanford NLP Group. [n.d.]. Stanford CoreNLP âĂŞ Natural language software. https://stanfordnlp.github.io/CoreNLP/

[32] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, et al. 2018. Achieving Human Parity on Automatic Chinese to English News Translation. *arXiv preprint arXiv:1803.05567* (2018).

[33] Jens Henriksson, Christian Berger, Markus Borg, Lars Tornberg, Cristofer Englund, Sankar Raman Sathyamoorthy, and Stig Ursing. 2019. Towards Structured Evaluation of Deep Neural Network Supervisors. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*.

[34] Jiuan-An Hsu. 2014. Error Classification of Machine Translation A Corpus-based Study on Chinese-English Patent Translation. *Translation Studies Quarterly* (2014).

[35] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

[36] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial Example Generation with Syntactically Controlled Paraphrase Networks. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

[37] Robin Jia and Percy Liang. 2017. Adversarial Examples for Evaluating Reading Comprehension Systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[38] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. 2018. Adversarial Logit Pairing. *arXiv preprint arXiv:1803.06373* (2018).

[39] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*.

[40] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980

[41] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. 2015. From Word Embeddings to Document Distances. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. 957–966.

[42] Fred. Lambert. 2016. Understanding the fatal Tesla accident on Autopilot and the NHTSA probe. https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/

[43] Vu Le, Mehrdad Afshari, and Zhendong Su. 2014. Compiler Validation via Equivalence Modulo Inputs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[44] Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10 (1966), 707–710. Issue 8.

[45] Sam Levin. 2018. Tesla fatal crash: 'autopilot' mode sped up car before driver killed, report finds. https://www.theguardian.com/technology/2018/jun/07/tesla-fatal-crash-silicon-valley-autopilot-mode-report

[46] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*.

[47] Christopher Lidbury, Andrei Lascu, Nathan Chong, and Alastair F. Donaldson. 2015. Many-Core Compiler Fuzzing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[48] Chin-Yew Lin. 2004. Rouge: A Package for Automatic Evaluation of Summaries. *Text Summarization Branches Out* (2004).

[49] Ji Lin, Chuang Gan, and Song Han. 2019. Defensive Quantization: When Efficiency Meets Robustness. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

[50] Mikael Lindvall, Dharmalingam Ganesan, Ragnar ÃĄrdal, and Robert E. Wiegand. 2015. Metamorphic Model-based Testing Applied on NASA DAT-an experience report. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*.

[51] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *CoRR* abs/1508.04025 (2015). arXiv:1508.04025 http://arxiv.org/abs/1508.04025

[52] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[53] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Wang Yadong. 2018. Deepgauge: Multi-Granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*.

[54] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *Proceedings of the 29th International Symposium on Software Reliability Engineering (ISSRE)*.

[55] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[56] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. 2019. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*.

[57] Fiona Macdonald. 2015. The Greatest Mistranslations Ever. http://www.bbc.com/culture/story/20150202-the-greatest-mistranslations-ever

[58] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

[59] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshops*.

[60] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)*.

[61] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2017. Adversarial Training Methods for Semi-Supervised Text Classification. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

[62] Pramod K. Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. 2018. Did the Model Understand the Question?. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.

[63] Christian Murphy, Gail E. Kaiser, Lifeng Hu, and Leon Wu. 2008. Properties of Machine Learning Applications for Use in Metamorphic Testing. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*.

[64] Arika Okrent. 2016. 9 Little Translation Mistakes That Caused Big Problems. http://mentalfloss.com/article/48795/9-little-translation-mistakes-caused-big-problems

[65] Thuy Ong. 2017. Facebook apologizes after wrong translation sees Palestinian man arrested for posting 'good morning'. https://www.theverge.com/us-world/2017/10/24/16533496/facebook-apology-wrong-translation-palestinian-arrested-post-good-morning

[66] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. In *IEEE Symposium on Security and Privacy*.

[67] 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 (ACL)*.

[68] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*.

[69] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. 2019. CRADLE: cross-backend validation to detect and localize bugs in deep learning libraries. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*.

[70] Michael Pradel and Koushik Sen. 2018. DeepBugs: A Learning Approach to Name-based Bug Detection. *Proceedings of the ACM on Programming Languages* OOPSLA (2018).

[71] Yao Qin, Nicholas Carlini, Ian Goodfellow, Garrison Cottrell, and Colin Raffel. 2018. Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*.

[72] 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 (ACL)*.

[73] Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-CortÃĺs. 2016. A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering (TSE)* 42 (2016). Issue 9.

[74] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

[75] Yikang Shen, Shawn Tab, Alessandro Sordoni, and Aaron Courville. 2019. Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

[76] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS)*.

[77] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. 2018. Attacks Meet Interpretability: Attribute-steered Detection of Adversarial Samples. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*.

[78] Wilson L. Taylor. 1953. "Cloze Procedure": A New Tool for Measuring Readability. *Journalism Bulletin* 30, 4 (1953), 415–433.

[79] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*.

[80] Tree. 2013. Adventures in Mistranslation: HSBC's Call to "Do Nothing". https://contentequalsmoney.com/mistranslation-hsbcs-call-to-do-nothing/

[81] Barak Turovsky. 2016. Ten years of Google Translate. https://blog.google/products/translate/ten-years-of-google-translate/

[82] Twitter. 2019. About Tweet translation. https://help.twitter.com/en/using-twitter/translate-tweets

[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, and Illia Kaiser, Lukasz abd Polosukhin. 2017. Attention is All you Need. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*.

[84] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial Sample Detection for Deep Neural Network through Model Mutation Testing. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*.

[85] WMT. [n.d.]. WMT Datasets. http://statmt.org/wmt17/translation-task.html

[86] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation. *arXiv preprint arXiv:1609.08144* (2016).

[87] Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. 2011. Testing and Validating Machine Learning Classifiers by Metamorphic Testing. *Journal of Systems and Software (JSS)* 84 (2011). Issue 4.

[88] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*.

[89] Chong Xiong, Charles R. Qi, and Bo Li. 2019. Generating 3D Adversarial Point Clouds. In *Proceedings of the 2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[90] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS)*.

[91] Dawei Yang, Chaowei Xiao, Bo Li, Jia Deng, and Mingyan Liu. 2019. Realistic Adversarial Examples in 3D Meshes. In *Proceedings of the 2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[92] Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating Neural Transformer via an Average Attention Network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.

[93] 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 (ASE)*.

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

[95] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. Deeproad: Gan-Based Metamorphic Autonomous Driving System Testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*.

[96] Wujie Zheng, Wenyu Wang, Dian Liu, Changrong Zhang, Qinsong Zeng, Yuetang Deng, Wei Yang, Pinjia He, and Tao Xie. 2018. Testing Untestable Neural Machine Translation: An Industrial Case. *arXiv preprint arXiv:1807.02340* (2018).

[97] Zhi Quan Zhou and Liqun Sun. 2018. Metamorphic Testing for Machine Translations: MT4MT. In *Proceedings of the 25th Australasian Software Engineering Conference (ASWEC)*.

[98] Zhi Quan Zhou, Shaowen Xiang, and Tsong Yueh Chen. 2016. Metamorphic Testing for Software Quality Assessment: A Study of Search Engines. *IEEE Transactions on Software Engineering (TSE)* 42 (2016). Issue 3.

[99] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 434–443.

[100] Chris. Ziegler. 2016. A Google self-driving car caused a crash for the first time. https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report