Testing Machine Translation via Referential Transparency

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*—Machine translation software has seen rapid progress in recent years due to the advancement of deep neural networks. People routinely use machine translation software in their daily lives, such as ordering food in a foreign restaurant, receiving medical diagnosis and treatment from foreign doctors, and reading international political news online. However, due to the complexity and intractability of the underlying neural networks, modern machine translation software is still far from robust and can produce poor or incorrect translations; this can lead to misunderstanding, financial loss, threats to personal safety and health, and political conflicts. To address this problem, we introduce *referentially transparent inputs (RTIs)*, a simple, widely applicable methodology for validating machine translation software. A referentially transparent input is a piece of text that should have similar translations when used in different contexts. Our practical implementation, *Purity*, detects when this property is broken by a translation. To evaluate RTI, we use *Purity* to test Google Translate and Bing Microsoft Translator with 200 unlabeled sentences, which detected 123 and 142 erroneous translations with high precision (79.3% and 78.3%). The translation errors are diverse, including examples of under-translation, over-translation, word/phrase mistranslation, incorrect modification, and unclear logic.**

***Index Terms*—Testing, Machine translation, Referential transparency, Metamorphic testing.**

I. Introduction

Machine translation software aims to fully automate translating text from a source language into a target language. In recent years, the performance of machine translation software has been improved significantly because of the development of neural machine translation (NMT) models [1]–[3]. In particular, machine translation software (*e.g.*, Google Translate [4] and Bing Microsoft Translator [5]) is approaching human-level performance in terms of human evaluation. More and more people are routinely employing machine translation in their daily lives, such as reading news and textbooks in foreign languages, communicating while traveling abroad, and conducting international trade. In 2016, Google Translate attracted more than 500 million users and translated more than 100 billion words per day [6]. Additionally, NMT models have been embedded in various software applications, such as Facebook [7] and Twitter [8].

Similar to traditional software (*e.g.*, a Web server), machine translation software's reliability is of great importance. Yet, modern translation software has been shown to return erroneous translations, leading to misunderstanding, financial loss, threats to personal safety and health, and political conflicts [9]–[14]. This behavior can be attributed to the brittleness of neural network-based systems, which is exemplified in autonomous car software [15], [16], sentiment analysis tools [17]–[19], and speech recognition services [20], [21]. Likewise, NMT models can be fooled by adversarial examples (*e.g.*, perturbing characters in the source text [22]) or natural noise (*e.g.*, typos [23]). The inputs generated by these approaches are mostly illegal, that is, they contain lexical (*e.g.*, "bo0k") or syntax errors (*e.g.*, "he home went"). However, inputs to machine translation software are generally lexically and syntactically correct. For example, Tencent, the company developing WeChat, a messaging app with more than one billion monthly active users, reported that its embedded NMT model can return erroneous translations even when the input is free of lexical and syntax errors [24].

There remains a dearth of automated testing solutions for machine translation software–at least in part because the problem is quite challenging. First, most of the existing parallel corpora have already been employed in the model training process. Thus, testing oracles of high quality are lacking. Second, in contrast to traditional software, the logic of neural machine translation software is largely embedded in the structure and parameters of the backend model. Thus, existing code-based testing techniques cannot directly be applied to testing NMT. Third, existing testing approaches for AI (artificial intelligence) software [15], [17]–[19], [25] mainly target much simpler use cases (*e.g.*, 10-class classification) and/or with clear oracles [26], [27]. In contrast, testing the correctness of translations is a more complex task: source text could have multiple correct translations and the output space is magnitudes larger. Last but not least, existing machine translation testing techniques [28], [29] generate test cases (*i.e.*, synthesized sentences) by replacing one word in a sentence via language models. Thus, their performance is limited by the proficiency of existing language models.

We introduce *RTIs (referentially transparent inputs)*, a novel and general concept, as a method for validating machine translation software. The core idea of RTI is inspired by *referential transparency* [30], [31], a concept in programming languages (specifically functional programming): a method

An RTI pair	Translations	Translation meanings	
Holmes in **a movie based on Bad Blood**	福尔摩斯在 电影的基础上坏血 (by Bing)	Holmes' **blood becomes bad based on a movie**	✗
a movie based on Bad Blood	一部基于坏血的电影 (by Bing)	a movie based on Bad Blood	✓

Fig. 1. Example of a referentially transparent input pair. The underlined phrase in the left column is an RTI extracted from the sentence. The differences in the translations are highlighted in red and their meanings are given in the right column. This RTI pair and its translations were reported by our approach as a suspicious issue. The first translation is erroneous.

should always return the same value for a given argument. In this paper, we define a *referentially transparent input (RTI)* as a piece of text that should have similar translations in different contexts. For example, "a movie based on Bad Blood" in Fig. 1 is an RTI. The key insight is to generate a pair of texts that contain the same RTI and check whether its translations in the pair are similar. To realize this concept, we implement *Purity*, a tool that extracts phrases from arbitrary text as RTIs. Specifically, given unlabeled text in a source language, *Purity* extracts phrases via a constituency parser [32] and constructs *RTI pairs* by grouping an RTI with either its containing sentence or a containing phrase. If a large difference exists between the translations of the same RTI, we report this pair of texts and their translations as a suspicious issue. The overview of *Purity* is illustrated in Fig. 2, where three RTI pairs are generated and one suspicious issue is reported. We will introduce the details of *Purity* along with Fig. 2 in Section III. The key idea of this paper is conceptually different from existing approaches [28], [29], which replace a word (*i.e.*, the context is fixed) and assume that the translation should have only small changes. In contrast, this paper assumes that the translation of an RTI should be similar across different sentences/phrases (*i.e.*, the context is varied).

We apply *Purity* to test Google Translate [33] and Bing Microsoft Translator [34] with 200 sentences crawled from CNN by He *et al*. [28]. *Purity* successfully reports 154 erroneous translation pairs in Google Translate and 177 erroneous translation pairs in Bing Microsoft Translator with high precision (79.3% and 78.3%), revealing 123 and 142 erroneous translations respectively.[1] The translation errors found are diverse, including under-translation, over-translation, word/phrase mistranslation, incorrect modification, and unclear logic. Compared with the state-of-the-art [28], [29], *Purity* can report more erroneous translations with higher precision. Due to its conceptual difference, *Purity* can reveal many erroneous translations that have not been found by existing approaches (illustrated in Fig. 6). Additionally, *Purity* spent 12.74s and 73.14s on average for Google Translate and Bing Microsoft Translator respectively, achieving comparable efficiency to the state-of-the-art methods. All the erroneous translations found are released [35] for independent validation. The source codes will also be released for reuse. The main contributions of this paper are as follows:

- The introduction of a novel, widely-applicable concept, *referentially transparent input (RTI)*, for systematic machine translation validation,
- A realization of RTI, *Purity*, that adopts a constituency parser to extract phrases and a bag-of-words (BoW) model to represent translations, and
- Empirical results demonstrating the effectiveness of RTI: based on 200 unlabeled sentences, *Purity* successfully found 123 erroneous translations in Google Translate and 142 erroneous translations in Bing Microsoft Translator with 79.3% and 78.3% precision, respectively.

II. Preliminaries

A. Referential Transparency

In the programming language field, referential transparency refers to the ability of an expression to be replaced by its corresponding value in a program without changing the result of the program [30], [31]. For example, mathematical functions (*e.g.*, square root function) are referentially transparent, while a function that prints a timestamp is not.

Referential transparency has been adopted as a key feature by functional programming because it allows the compiler to reason about program behavior easily, which further facilitates higher-order functions (*i.e.*, a series of functions can be glued together) and lazy evaluation (*i.e.*, delay the evaluation of an expression until its value is needed) [36]. The terminology "referential transparency" is used in a variety of fields with different meanings, such as logic, linguistics, mathematics, and philosophy. Inspired by the referential transparency concept in functional programming, a metamorphic relation can be defined within an RTI pair.

B. Metamorphic Relation

Metamorphic relations are necessary properties of functionalities of the software under test. In metamorphic testing [37]–[39], the violation of a metamorphic relation will be suspicious and indicates a potential bug. We develop a metamorphic relation for machine translation software as follows: RTIs (e.g., noun phrases) should have similar translations in different contexts. Formally, for an RTI r, assume we have two different contexts C_1 and C_2 (i.e., different pieces of surrounding

[1]One erroneous translation could appear in multiple erroneous translation pairs (i.e., erroneous issues).

words). $C_1(r)$ and $C_2(r)$, which form an RTI pair, are the pieces of text containing r and the two contexts respectively. To test the translation software T, we could obtain their translations $T(C_1(r))$ and $T(C_2(r))$. The metamorphic relation is defined as:

$$dist_r(T(C_1(r)), T(C_1(r))) \leq d, \quad (1)$$

where $dist_r$ denotes the distance between the translations of r in $T(C_1(r))$ and in $T(C_2(r))$; d is a threshold controlled by the developers. In the following section, we will introduce our approach in detail with an example (Fig. 2).

III. RTI AND *Purity*'S IMPLEMENTATION

This section introduces *referentially transparent inputs (RTIs)* and our implementation, *Purity*. An RTI is defined as a piece of text that has similar translations across texts (*e.g.*, sentences and phrases). Given a sentence, our approach intends to find its RTIs—phrases in the sentence that exhibit referential transparency—and utilize them to construct test inputs. To realize RTI's concept, we implement a tool called *Purity*. The input of *Purity* is a list of unlabeled, monolingual sentences, while its output is a list of suspicious issues. Each issue contains two pairs of text: a base phrase (*i.e.*, an RTI) and its container phrase/sentence, and their translations. Note that *Purity* should detect errors in the translation of either the base or container text. Fig. 2 illustrates the process used by *Purity*, which has the following four steps:

1) *Identifying referentially transparent inputs.* For each sentence, we extract a list of phrases as its RTIs by analyzing the sentence constituents.
2) *Generating pairs in source language.* We pair each phrase with either a containing phrase or the original sentence to form RTI pairs.
3) *Collecting pairs in target language.* We feed the RTI pairs to the machine translation software under test and collect their corresponding translations.
4) *Detecting translation errors.* In each pair, the translations of the RTI pair are compared with each other. If there is a large difference between the translations of the RTI, *Purity* reports the pair as potentially containing translation error(s).

Algorithm 1 shows the pseudo-code of our RTI implementation, which will be explained in detail in the following sections.

Algorithm 1 RTI implemented as *Purity*.

Require: $source\_sents$: a list of sentences in source language
d: the distance threshold
Ensure: $suspicious\_issues$: a list of suspicious pairs

```
suspicious_issues ← List( )                    ▷ Initialize with empty list
for all source_sent in source_sents do
    constituency_tree ← PARSE(source_sent)
    head ← constituency_tree.head( )
    RTI_source_pairs ← List( )
    RECURSIVENPFINDER(head, List( ), RTI_source_pairs)
    RTI_target_pairs ← TRANSLATE(RTI_source_pairs)
    for all target_pair in RTI_target_pairs do
        if DISTANCE(target_pair) > d then
            Add source_pair, target_pair to suspicious_issues
return suspicious_issues

function RECURSIVENPFINDER(node, rtis, all_pairs)
    if node is leaf then
        return
    if node.constituent is NP then
        phrase ← node.string
        for all container_phrase in rtis do
            Add container_phrase, phrase to all_pairs
        Add phrase to rtis
    for all child in node.children( ) do
        RECURSIVENPFINDER(child, rtis.copy( ), all_pairs)
    return all_pairs

function DISTANCE(target_pair)
    rti_BOW ← BAGOFWORDS(target_pair[0])
    container_BOW ← BAGOFWORDS(target_pair[1])
    return |rti_BOW \ container_BOW|
```

A. Identifying RTIs

In order to collect a list of RTIs, we must find pieces of text with unique meaning, *i.e.* their meaning should hold across contexts. To guarantee the lexical and syntactic correctness of RTIs, we extract them from published text (*e.g.*, sentences in Web articles).

Specifically, *Purity* extracts noun phrases from a set of sentences in a source language as RTIs. For example, in Fig. 2, the phrase "chummy bilateral talks" will be extracted; this phrase should have similar translations when used in different sentences (*e.g.*, "I attended chummy bilateral talks." and "She held chummy bilateral talks.") For the sake of simplicity and to avoid grammatically strange phrases, we only consider noun phrases in this paper.

We identify noun phrases using a constituency parser, a readily available natural language processing (NLP) tool. A constituency parser identifies the syntactic structure of a piece of text, outputting a tree where the non-terminal nodes are constituency relations and the terminal nodes are the words (example shown in Fig. 3). To extract all noun phrases, we traverse the constituency parse tree and pull out all the *NP* (noun phrase) relations.

Note that in general, an RTI can contain another shorter RTI. For example, the second RTI pair in Fig. 1 contains two RTIs: "Holmes in a movie based on Bad Blood" is the containing RTI to "a movie based on Bad Blood" This holds true when noun phrases are used as RTIs as well, since noun phrases can contain other noun phrases.

Once we have obtained all the noun phrases from a sentence, we filter out those containing more than 10 words and those containing less than 3 words[2] that are not stop-words.[3] This filtering helps us concentrate on unique phrases that are more likely to carry a single meaning and greatly reduces false

[2]These filter values were tuned empirically.

[3]A stop-word is a common word such as "is", "this", "an", etc.

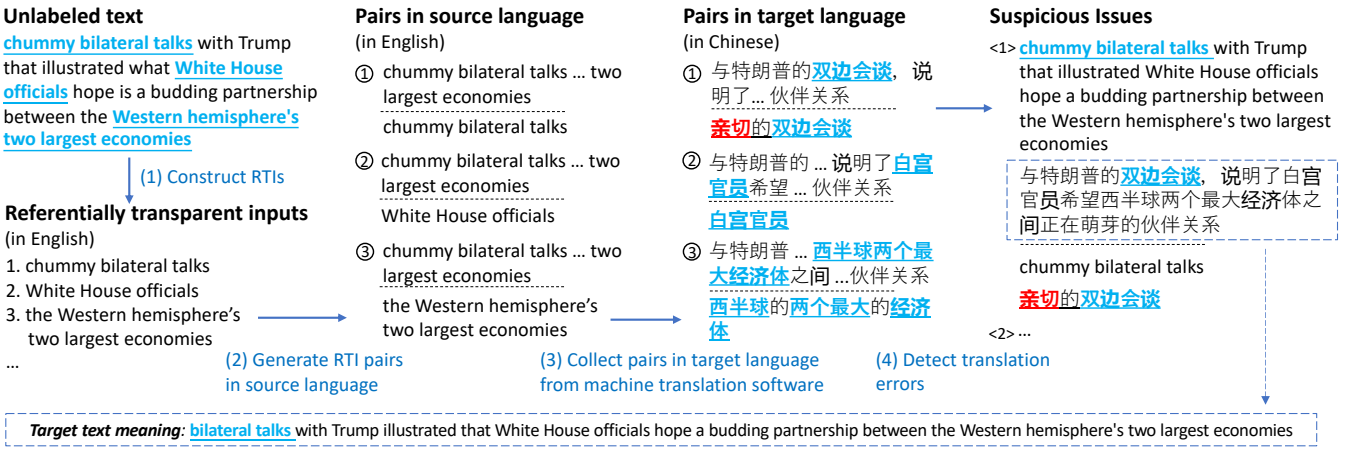


Fig. 2. Overview of our RTI implementation. We use one English phrase as input for clarity and simplicity.

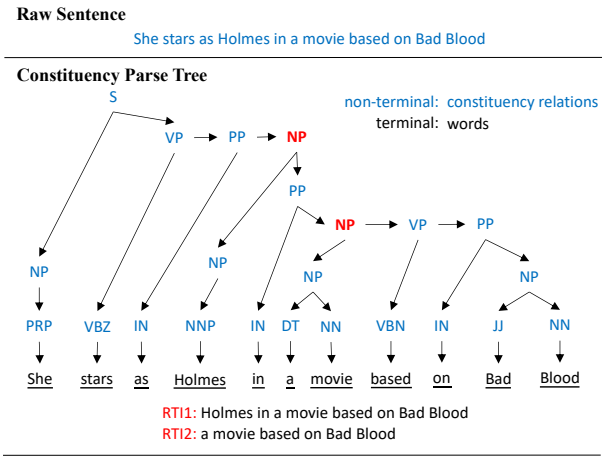


Fig. 3. A constituency parse tree example. The non-terminal nodes in bold and red are the RTIs extracted by our approach.

positives. The remaining noun phrases are regarded as RTIs in *Purity*.

B. Generating Pairs in Source Language

Once a list of RTIs has been generated, each must be paired with containing phrases, which will be used for referential transparency validation (Section III-D). Specifically, each RTI pair should have two different pieces of text that contain the same phrase. To generate these pairs, we pair an RTI with the full text in which it was found (as in Fig. 2) and with all the containing RTIs (*i.e.*, other containing noun phrases) from the same sentence. For example, as illustrated in Fig. 3, two RTIs are found; note that "Holmes in a movie based on Bad Blood" is the containing RTI to "a movie based on Bad Blood". Thus, 3 RTI pairs will be constructed: (1) RTI1 and the original sentence; (2) RTI2 and the original sentence; and (3) RTI1 and RTI2.

C. Collecting Pairs in Target Language

The next step is to input RTI pairs (in the given source language) to the machine translation software under test and collect their translations (in any chosen target language). We use the APIs provided by Google and Bing in our implementation, which return results identical to Google Translate and Bing Microsoft Translator's Web interfaces [33], [34].

D. Detecting Translation Errors

Finally, in order to detect translation errors, translated pairs from the previous step are checked for RTI invariance. Detecting the absensce of an RTI in a translation while avoiding false positives is non-trivial. For example, in Fig. 2, the RTI in the first pair is "chummy bilateral talks." Given the Chinese translation of the whole original sentence, it is difficult to identify which characters refer to the RTI. Words may be reordered while preserving the inherent meaning, so exact matches between RTI and container translations are not guaranteed.

NLP techniques such as word alignment [40], [41], which maps a word/phrase in source text to a word/phrase in its target target, could be employed for this component of the implementation. However, performance of existing tools is poor and runtime can be quite slow. Instead, we adopt a bag-of-words (BoW) model, a representation that only considers the appearance(s) of each word in a piece of text (see Fig. 4 for example). Note that this representation is a multiset. While the BoW model is simple, it has proven quite effective for modeling text in many NLP tasks. For *Purity*, using an n-gram representation of the target text provides similar performance.

BoW = {"we": *1*, "watched": *1*, "two": *2*, "movies": *1*,
"and": *1*, "basketball": *1*, "games": *1*}

Fig. 4. Bag-of-words representation of "we watched two movies and two basketball games."

Each translated pair consists of a translation of an RTI,

$T(r)$,[4] and of its container $T(C_{con}(r))$. After obtaining the BoWs representation of both translations (BoW_r and BoW_{con}), the distance $dist_r(T(r), T(C_{con}(r)))$ is calculated by $dist(BoW_r, BoW_{con})$ as follows:

$$dist(BoW_r, BoW_{con}) = |BoW_r \setminus BoW_{con}| \quad (2)$$

In words, this metric measures how many word occurrences are in $T(r)$ but not in $T(C_{con}(r))$. For example, the distance between "we watch two movies and two basketball games" ($T(C_{con}(r))$) and "two interesting books" ($T(r)$) is 2. If the distance is larger than a threshold d, which is a chosen hyperparameter, the translation pair and their source texts will be reported by our approach as a suspicious issue, indicating that at least one of the translations may contain errors. For example, in the suspicious issue in Fig. 2, the distance is 2 because Chinese characters 亲切 do not appear in the translation of the container $T(C_{con}(r))$.[5]

We note that theoretically, this implementation cannot detect over-translation errors in $T(C_{con}(r))$ because additional word occurrence in $T(C_{con}(r))$ will not change the distance as calculated in Equ. 2. However, this problem does not often occur since the source text $C_{con}(r)$ is frequently the RTI in another RTI pair, in which case over-translation errors can be detected in the latter RTI pair.

IV. Evaluation

In this section, we evaluate the performance of *Purity* by applying it to Google Translate and Bing Microsoft Translator. Specifically, this section aims at answering the following research questions:

- RQ1: How precise is the approach at finding erroneous issues?
- RQ2: How many erroneous translations can our approach report?
- RQ3: What kinds of translation errors can our approach find?
- RQ4: How efficient is the approach?

A. Experimental Setup and Dataset

a) Experimental environments: 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. For sentence parsing, we use the shift-reduce parser by Zhu *et al.* [32], which is implemented in Stanford's CoreNLP library.[6] Our experiments consider the English→Chinese language setting because of the knowledge background of the authors.

[4]In our implementation, the context could be an empty string. Thus, $C_{empty}(r) = r$.

[5]For Chinese text, *Purity* regards each character as a word.

[6]https://stanfordnlp.github.io/CoreNLP/

TABLE I
Statistics of input sentences for evaluation. Each corpus contains 100 sentences.

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

b) Comparison: We compare *Purity* with two state-of-the-art approaches: SIT [28] and TransRepair (ED) [29]. We obtained the source code of SIT from the authors. The authors of TransRepair could not release their source code due to industrial confidentiality. Thus, we carefully implement their approach following descriptions in the paper and consulting the work's main author for crucial implementation details. TransRepair uses a threshold of 0.9 for the cosine distance of word embeddings to generate word pairs. In our experiment, we use 0.8 as the threshold because we were unable to reproduce the quantity of word pairs that the paper reported using 0.9. In this paper, we evaluate TransRepair-ED because it achieves the highest precision among the four metrics on Google Translate and better overall performance than TransRepair-LCS for Transformers (Table 2 of [29]). In addition, we re-tune the parameters of SIT and TransRepair using the strategies introduced in their papers. All the approaches in this evaluation are implemented in Python and will be released for reuse.

c) Dataset: Purity tests machine translation software with lexically- and syntactically-correct real-world sentences. We use the dataset collected from CNN articles released by He *et al.* [28]. The details of this dataset are illustrated in Table I. This dataset contains two corpora: *Politics* and *Business*. Sentences in the "Politics" dataset contains 4∼32 words (the average is 19.2) and they contain 1,918 words and 933 non-repetitive words in total. We use corpora from both categories to evaluate the performance of *Purity* on sentences with different terminology.

B. Precision on Finding Erroneous Issues

Our approach automatically reports suspicious issues that contain inconsistent translations on the same RTI. Thus, the effectiveness of the approach lies in two aspects: (1) how precise are the reported issues; and (2) how many erroneous translations can *Purity* find? In this section, we evaluate the precision of the reported pairs, *i.e.*, how many of the reported issues contain real translation errors. Specifically, we apply *Purity* to test Google Translate and Bing Microsoft Translator using the datasets characterized by Table I. To verify the results, two authors manually inspect all the suspicious issues separately and then collectively decide (1) whether an issue contains translation error(s); and (2) if yes, what kind of translation error it contains.

1) Evaluation Metric: The output of *Purity* is a list of suspicious issues, each containing (1) an RTI, r, in source

language and its translation, $T(r)$; and (2) a piece of text in a source language, which contains the RTI, $C_{con}(r)$, and its translation, $T(C_{con}(r))$. We define the precision as the percentage of pairs that have translation error(s) in $T(r)$ or $T(C_{con}(r))$. Explicitly, for a suspicious issue p, we set $error(p)$ to $true$ if $T_p(r)$ or $T_p(C_{con}(r))$ has translation error(s) (*i.e.*, when the suspicious issue is an *erroneous issue*). Otherwise, we set $error(p)$ to $false$. Given a list of suspicious issues, the precision is calculated by:

$$\text{Precision} = \frac{\sum_{p \in P} \mathbb{1}\{error(p)\}}{|P|}, \tag{3}$$

where P is the suspicious issues returned by *Purity* and $|P|$ is the number of the suspicious issues.

2) Results: The results are presented in Table II. We observe that if the goal is to find as many issues as possible (*i.e.*, $d = 0$), *Purity* achieves 78%∼79.8% precision while reporting 67∼99 erroneous issues. For example, when testing Bing Microsoft Translator with the "Business" dataset, *Purity* reports 100 suspicious issues, while 78 of them contain translation error(s), leading to 78% precision. If we want *Purity* to be more accurate, we can use a larger distance threshold. For example, when we set the distance threshold to 5, *Purity* achieves 100% precision on all experimental settings. Note the precision does not increase monotonically with the threshold value. For "Bing-Politics," the precision drops 1.9% when changing the threshold value from 2 to 3. So although the number of false positives decreases, the number of true positives may decrease as well. In our comparisons, we find *Purity* detects more erroneous issues with higher precision compared with all the existing approaches. To compare with SIT, we focus on the top-1 results (*i.e.* the translation that is most likely to contain errors) reported by their system. In particular, the top-1 output of SIT contains (1) the original sentence and its translation and (2) the top-1 generated sentence and its translation. For direct comparison, we regard the top-1 output of SIT as a suspicious issue. TransRepair reports a list of suspicious sentence pairs and we regard each reported pair as a suspicious issue. Equ. 3 is used to compute the precision of the compared approaches. The results are presented in the right-most columns of Table II.

When the distance threshold is at its lowest (*i.e.*, $d = 0$), *Purity* finds more erroneous issues with higher precision compared with SIT and TransRepair. For example, when testing Google Translate on the "Politics" dataset, *Purity* finds 87 erroneous issues with 79.8% precision, while SIT only finds 34 erroneous issues with 65.3% precision. When $d = 2$, *Purity* detects a similar number of erroneous issues to SIT but with significantly higher precision. For example, when testing Bing Microsoft Translator on the "Politics" dataset, *Purity* finds 39 erroneous issues with 92.8% precision, while SIT finds 36 erroneous issues with 70.5% precision.[7] Although the precision comparison is not apples-to-apples, we believe the results have shown the superiority of *Purity*. As real-world source sentences are almost unlimited, in practice, we could set $d = 2$ for this language setting to obtain a decent amount of erroneous issues with high precision.

We believe *Purity* achieves a much higher precision because of the following reasons. First, existing approaches rely on pre-trained models (*i.e.*, BERT [42] for SIT and GloVe [43] and spaCy [44] for TransRepair) to generate sentences pairs. Although BERT should do well on this task, it could generate sentences of strange semantics, leading to false positives. Differently, *Purity* directly extract phrases from real-world sentences to construct RTI pairs and thus does not have such kind of false positives. In addition, SIT relies on dependency parsers [45] in target sentence representation and comparison. The dependency parser could return incorrect dependency parse trees, leading to false positives.

Source text	Target text
a lot of innovation coming from other parts of the world	很多来自世界其他地方的创新 (by Bing)
innovation coming from other parts of the world	来自世界其他地区的创新 (by Bing)
The South has emerged as a hub of new auto manufacturing by foreign makers thanks to lower manufacturing costs and less powerful unions.	由于较低的制造成本和较弱的工会，南方已成为外国制造商新汽车制造的枢纽。(by Google)
foreign makers thanks	外国厂商谢谢 (by Google)
He was joined by Justices Ruth Bader Ginsburg, Elena Kagan and Sonia Sotomayor.	鲁思·巴德尔·金斯堡法官、埃琳娜·卡根法官和索尼娅·索托马约尔法官也加入了他的行列。(by Bing)
Justices Ruth Bader Ginsburg	法官露丝·巴德尔·金斯堡 (by Bing)

Fig. 5. False positive examples.

3) False Positives: False positives of *Purity* come from three sources. In Fig. 5, we present false positive examples when $d = 0$. First, a phrase could have multiple correct translations. As shown in the first example, "parts" have two correct translations (*i.e.*, 地方 and 地区) in the context "other parts of the word". However, when $d = 0$, it will be reported. This category accounts for most of *Purity*'s false positives. To alleviate this kind of false positive, we could tune the distance threshold d or maintain an alternative translation dictionary. Second, the constituency parser that we use to identify noun phrases could return a non-noun phrase. In the second example, "foreign makers thanks" is identified as a noun phrase, which leads to the change of phrase meaning. In our experiments, 6 false positives are caused by incorrect output from the constituency parser. Third, proper names are often transliterated and thus could have different correct results. In the third example, the name "Ruth" has two correct transliterations, leading to a false positive. In our experiments 1 false positive is caused by the transliteration of proper names.

C. Erroneous Translation

We have shown that *Purity* can report erroneous issues with high precision, where each erroneous issue contains

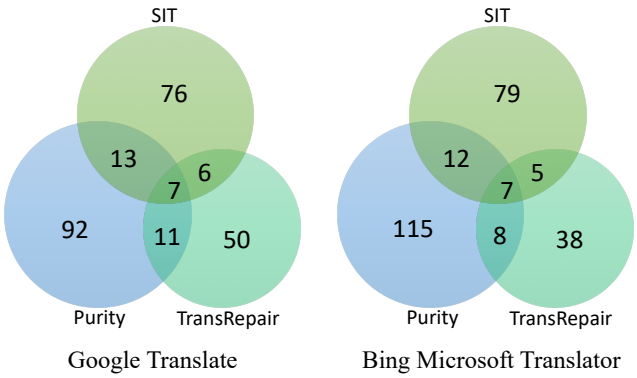
[7]Note the precision results are different from those reported by He *et al.* [28] because Google Translate and Bing Microsoft Translator continuously update their model.

TABLE II
Purity'S PRECISION (# OF ERRONEOUS ISSUES/# OF SUSPICIOUS ISSUES) USING DIFFERENT THRESHOLD VALUES.

	Purity						SIT	TransRepair
	0	1	2	3	4	5		
Google-Politics	79.8% (87/109)	81.9% (59/72)	94.5% (35/37)	100% (18/18)	100% (11/11)	100% (7/7)	65.3% (34/52)	64.2% (45/70)
Google-Business	78.8% (67/85)	79.3% (46/58)	100% (21/21)	100% (5/5)	N.A.	N.A.	64.7% (33/51)	61.1% (22/36)
Bing-Politics	78.5% (99/126)	82.9% (68/82)	92.8% (39/42)	90.9% (20/22)	100% (7/7)	100% (3/3)	70.5% (36/51)	70.5% (24/34)
Bing-Business	78.0% (78/100)	80.0% (48/60)	90.9% (20/22)	90.0% (9/10)	83.3% (5/6)	100% (3/3)	62.7% (32/51)	55.0% (22/40)

TABLE III
THE NUMBER OF TRANSLATIONS THAT CONTAIN ERRORS USING DIFFERENT THRESHOLD VALUES.

	Purity						SIT	Trans Repair
	0	1	2	3	4	5		
Google-Politics	69	53	38	24	15	9	50	44
Google-Business	54	39	20	8	0	0	52	30
Bing-Politics	74	56	42	22	8	4	55	33
Bing-Business	68	46	20	9	6	5	48	25



Google Translate Bing Microsoft Translator

Fig. 6. Erroneous translations reported by *Purity*, SIT, and TransRepair.

at least one erroneous translation. Thus, to further evaluate the effectiveness of *Purity*, in this section, we study how may erroneous translations *Purity* can find. Specifically, if an erroneous translation appears in multiple erroneous issues, it will be counted once. Table III presents the number of erroneous translations under the same experimental settings as in Table II. We can observe that when $d = 0$, *Purity* found 54∼74 erroneous translations. If we intend to have a higher precision by setting a larger distance threshold, we will reasonably obtain fewer erroneous translations. For example, if we want to achieve 100% precision, we can obtain 32 erroneous translations in Google Translate ($d = 3$).

We further study the erroneous translations found by *Purity*, SIT and TransRepair. Fig. 6 demonstrates the results via Venn diagrams. We can observe that, 7 erroneous translations from Google Translate and 7 erroneous translations from Bing Microsoft Translator can be detected by all the three approaches. These are the translations for some of the original source sentences. 207 erroneous translations are unique to *Purity* while 155 erroneous translations are unique to SIT and 88 erroneous translations are unique to TransRepair. After inspecting all the erroneous translations, we find that *Purity* is effective at reporting translation errors for phrases. Meanwhile, the unique errors to SIT are mainly from similar sentence of one noun or adjective difference. The unique errors to TransRepair mainly come from similar sentences of one number difference (*e.g.*, "five" → "six"). Based on these results, we believe our approach complements the state-of-the-art approaches.

D. Types of Reported Translation Errors

Purity is capable of detecting translation errors of diverse kinds. Specifically, in our evaluation, *Purity* has successfully detected 5 kinds of translation errors: under-translation, over-translation, word/phrase mistranslation, incorrect modification, and unclear logic. Table IV presents the number of translations that have a specific kind of error. We can observe that word/phrase mistranslation and unclear logic are the most common translation errors.

TABLE IV
NUMBER OF TRANSLATIONS THAT HAVE SPECIFIC ERRORS IN EACH CATEGORY.

	Under translation	Over translation	Word/phrase mistranslation	Incorrect modification	Unclear logic
Google-Politics	17	9	43	5	12
Google-Business	12	6	29	8	11
Bing-Politics	8	2	51	4	23
Bing-Business	11	5	38	6	32

To provide a glimpse of the diversity of the uncovered errors, this section highlights examples of all the 5 kinds of errors. The variety of the detected translation errors demonstrates RTI's (offered by *Purity*) efficacy and broad applicability. We align the definition of these errors with SIT [28] because it is the first work that found and reported these 5 kinds of translation errors.

1) Under-translation: If some parts of the source text are not translated in the target text, it is an under-translation error. For example, in Fig. 7, "magnitude of" is not translated by Google Translate. Under-translation often leads to target

sentences of different semantic meanings and the lack of crucial information. Fig. 2 also reveals an under-translation error. In this example, the source text emphasizes that the bilateral talks are chummy while this key information is missing in the target text.

Source	the sorts of problems we work on and the almost anxiety provoking magnitude of data with which we get to work
Target	我们正在研究的各种问题以及几乎令人焦虑的数据 (by Google)
Target meaning	the sorts of problems we work on and the almost anxiety provoking data with which we get to work

Fig. 7. Example of under-translation error detected.

2) Over-translation: If some parts of the target text are not translated from word(s) of the source text or some parts of the source text are unnecessarily translated for multiple times, it is an over-translation error. In Fig. 8, "was an honor" is translated twice by Google Translate in the target text while it only appears once in the source text, so it is an over-translation error. Over-translation brings unnecessary information and thus can easily cause misunderstanding.

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
Target	荣幸地报道了该国首都的追悼会，然后前往得克萨斯州进行另一项服务以及葬礼列车，这是一种荣幸 (by Google)
Target meaning	It was an honor to cover a memorial service of the nation's capital and then traveling to Texas to conduct another service and a funeral train was an honor

Fig. 8. Example of over-translation error detected.

3) Word/phrase Mistranslation: If some words or phrases in the source text is incorrectly translated in the target text, it is a word/phrase mistranslation error. In Fig. 9, "creating housing" is translated to "building houses" in the target text. This error is caused by ambiguity of polysemy. The word "housing" means "a general place for people to live in" or "a concrete building consisting of a ground floor and upper storeys." In this example, the translator mistakenly thought "housing" refers to the later meaning, leading to the translation error. In addition to ambiguity of polysemy, word/phrase mistranslation can be also caused by the surrounding semantics. In the second example of Fig. 9, "plant" is translated to "company" in the target text. We think that in the training data of the NMT model, "General Motors" often has the translation "General Motors company", which leads to a word/phrase mistranslation error in this scenario.

4) Incorrect Modification: If some modifiers modify the wrong element, it is an incorrect modification error. In Fig. 10, "better suited for a lot of business problems" should modify "more specific skill sets". However, Bing Microsoft Translator inferred they are two separate clauses, leading to an incorrent modification error.

5) Unclear Logic: If all the words are correctly translated but the logic of the target text is wrong, it is an unclear logic error. In Fig. 11, Bing Microsoft Translator correctly translated "approval" and "two separate occasions". However, Bing Microsoft Translator returned "approve two separate occasions" instead of "approval on two separate occasions" because the translator does not understand the logical relation between them. Fig. 1 also demonstrates an unclear logic error. Unclear logic errors widely exist in the translations returned by modern machine translation software, which is to some extent a sign of whether the translator truely understands certain semantic meanings.

Source	Advertisers who are not creating housing , employment or credit ads
Target	未制作住房，就业或信用广告的广告客户 (by Google)
Target meaning	Advertisers who are not building houses , employment or credit ads
Source	the General Motors plant
Target	通用汽车公司 (by Bing)
Target meaning	the General Motors company

Fig. 9. Examples of word/phrase mistranslation errors detected.

Source	more specific skill sets that are better suited for a lot of business problems
Target	更具体的技能集，更适合于许多业务问题 (by Bing)
Target meaning	more specific skill sets, better suited for a lot of business problems

Fig. 10. Example of incorrect modification error detected.

Source	approval on two separate occasions
Target	批准两个不同的场合 (by Bing)
Target meaning	approve two separate occasions

Fig. 11. Example of unclear logic error detected.

E. Running Time

In this section, we study the efficiency (*i.e.*, running time) of *Purity*. Specifically, we adopt *Purity* to test Google Translate and Bing Microsoft Translator with the "Politics" and the "Business" dataset. For each experimental setting, we run *Purity* 10 times and use the average time as the final result. Table V presents the total running time of *Purity* as well as the detailed running time for initialization, RTI pairs construction, translation collection, and referential transparency violation detection.

We can observe that *Purity* spent less than 15 seconds on testing Google Translate and around 1 minute on testing Bing Microsoft Translator. Specifically, more than 90% of the time is used in the collection of translations via translators' APIs . In our implementation, we invoke the translator API once for each piece of source text and thus the network communication

TABLE V
RUNNING TIME OF *Purity* (SEC)

		Google Politics	Google Business	Bing Politics	Bing Business
Initialization	Purity	0.0048	0.0042	0.0058	0.0046
RTI construction		0.83	0.85	0.86	0.89
Translation		11.51	12.22	72.79	71.66
Detection		0.0276	0.0263	0.0425	0.0301
Total		12.38	13.10	73.70	72.59
SIT		391.83	365.22	679.65	631.26
TransRepair		15.17	12.71	56.39	54.24

time is included. If developers intend to test their own machine translation software with *Purity*, the running time of this step will be even less.

Table V also presents the running time of SIT and TransRepair using the same experimental settings. SIT spent more than 6 minutes to test Google Translate and around 11 minutes to test Bing Microsoft Translator. This is mainly because SIT translates 44,414 words for the "Politics" dataset and 41,897 words for the "Business" dataset. Meanwhile, *Purity* and TransRepair require fewer translations (7,565 and 6,479 for *Purity* and 4,271 and 4,087 for TransRepair). Based on these results, we conclude that *Purity* achieves comparable efficiency to the state-of-the-art methods.

F. Fine-tuning with Errors Reported by Purity

The ultimate goal of testing is to improve software robustness. Thus, in this section, we study whether reported mistranslations can act as a fine-tuning set to both improve the robustness of NMT models and quickly fix errors found during testing. Fine-tuning is a common practice in NMT since the domain of the target data (i.e. data used at runtime) is often different than that of the training data [46], [47]. To simulate this situation, we train a transformer network with global attention [3]—a standard architecture for NMT models—on the WMT'18 ZH–EN (Chinese-to-English) corpus [48], which contains $\sim$ 20M sentence pairs. We reverse the standard direction of translation (*i.e.* to EN–ZH) for comparison with our other experiments. We use the fairseq framework [49] to create the model.

To test our NMT model, we crawled the 10 latest articles under the "Entertainment" category of CNN website and randomly extract 80 English sentences. The dataset collection process aligns with that of the "Politics" and the "Business" datasets [28] used in the main experiments. We run *Purity* with the "Entertainment" dataset using our trained model as the system under test; *Purity* successfully finds 42 erroneous translations. We manually label them with correct translations and fine-tune the NMT model on these 42 translation pairs for 8 epochs—until loss on the WMT'18 validation set stops decreasing. After this fine-tuning, 40 of the 42 sentences are correctly translated. One of the two translations that were not corrected can be attributed to parsing errors; while the other (source text: "one for Best Director") has an "ambiguous reference" issue, which essentially makes it difficult to translate without context. Meanwhile, the BLEU score on the WMT'18 validation set stayed well within standard deviation [50]. This demonstrates that error reported by *Purity* can indeed be fixed without retraining a model from scratch – a resource and time intensive process.

V. DISCUSSIONS

A. RTI for Robust Machine Translation

In this section, we discuss the utility of RTI towards building robust machine translation software. Compared with traditional software, the error fixing process of machine translation software is arguably more difficult because the logic of NMT models lies within a complex model structure and its parameters rather than human-readable code. Even if the computation which causes a mistranslation can be identified, it is often not clear how to change the model to correct the mistake without introducing new errors. While model correction is a difficult open problem and is not the main focus of our paper, we find it important to explain that the translation errors found by *Purity* can be used to both fix and improve machine translation software.

For online translation systems, the fastest way to fix a mistranslation is to hard-code the translation pair. Thus, the translation errors found by RTI can be quickly and easily addressed by developers to avoid mistranslations that may lead to negative effects [9]–[14]. The more robust solution is to incorporate the mistranslation into the training dataset. In this case, a developer can add the source sentence of a translation error along with its correct translation to the training set of the neural network and retrain or fine-tune the network. While retraining a large neural network from scratch can take days, fine-tuning on a few hundred mistranslations takes only a few minutes, even for the large, SOTA models. We note that this method does not absolutely guarantee the mistranslation will be fixed, but our experiments (Section IV-F) show it to be quite effective in resolving errors. The developers may also find the reported issues useful for further analysis/debugging because it resembles debugging traditional software via input minimization/localization. In addition, as RTI's reported results are in pairs, they can be utilized as a dataset for future empirical studies on translation errors.

B. Change of Language

In our implementation, *Purity*, we use English as the source language and Chinese as the target language. To match our exact implementation, there needs to be a constituency parser—or data to train such a parser—available in the chosen source language, as this is how we find RTIs. The Stanford Parser[8]

[8]https://nlp.stanford.edu/software/lex-parser.html#Download

currently supports six languages. Alternatively, one can train a parser following, for example, Zhu *et al*. [32]. Other modules of *Purity* remain unchanged. Thus, in principle, it is quite easy to re-target RTI to other languages. Note that while we expect the RTI property to hold for most of the languages, there may be confounding factors in the structure of a language that break our assumptions.

VI. Related Work

A. Robustness of AI Software

Recently, Artificial Intelligence (AI) software has been adopted by many domains; this is largely due to the modelling abilities of deep neural networks. However, these systems can generate erroneous outputs that lead to fatal accidents [51]–[53]. To explore the robustness of AI software, a line of research has focused on attacking different systems that use deep neural networks, such as autonomous cars [25], [54] and speech recognition services [20], [55]. These work aim to fool AI software by feeding input with imperceptible perturbations (*i.e.*, adversarial examples). Meanwhile, researchers have also designed approaches to improve AI software's robustness, such as robust training mechanisms [56]–[58], adversarial examples detection approaches [59], [60], and testing/debugging techniques [15], [16], [61]–[66]. Our paper also studies the robustness of a widely-adopted AI software, but focuses on machine translation systems, which has not been explored by these papers. Additionally, most of these approaches are white-box, utilizing gradients/activation values, while our approach is black-box, requiring no model internal details at all.

B. Robustness of NLP Systems

Inspired by robustness studies in the computer vision field, NLP (natural language processing) researchers have started exploring attack and defense techniques for various NLP systems. Typical examples include sentiment analysis [17]–[19], [67], [68], textual entailment [18], and toxic content detection [19]. However, these are all basic classification tasks while machine translation software is more complex in terms of both model output and network structure.

The robustness of other complex NLP systems has also been studied in recent years. Jia and Liang [27] proposed a robustness evaluation scheme for the Stanford Question Answering Dataset (SQuAD), which is widely used in the evaluation of reading comprehension systems. They found that even the state-of-the-art system, achieving near human-level F1-score, fails to answer questions about paragraphs correctly when an adversarial sentence is inserted. Mudrakarta *et al.* [26] also generate adversarial examples for question answering tasks on images, tables, and passages of text. These approaches typically perturb the system input and assume that the output (*e.g.*, a person name or a particular year) should remain the same. However, the output of machine translation (*i.e.*, a piece of text) is more complex. In particular, one source sentence could have multiple correct target sentences. Thus, testing machine translation software, which is the goal of this paper, is more difficult.

C. Robustness of Machine Translation

Recently, researchers have started to explore the robustness of NMT models. Belinkov and Bisk [23] found that both synthetic (*e.g.*, character swaps) and natural (*e.g.*, misspellings) noise in source sentences could break character-based NMT models. In contrast, our approach aims at finding lexically- and syntactically-correct source texts that lead to erroneous output by machine translation software, which the errors more commonly found in practice. To improve the robustness of NMT models, various robust training mechanisms have been studied [69], [70]. In particular, noise is added to the input and/or the internal network embeddings during training. Different from these approaches, we focus on testing machine translation.

Zheng *et al.* [24] proposed specialized approaches to detect under- and over-translation errors respectively. Different from them, our approach aims at finding general errors in translation. He *et al*. [28] and Sun *et al*. [29] proposed metamorphic testing methods for general translation errors: they compare the translations of two similar sentences (*i.e.*, differed by one word) by sentence structures [28] and four existing metrics on sub-strings [29] respectively. In addition, Sun *et al*. [29] designed an automated translation error repair mechanism. Compared with these approaches, RTI can find more erroneous translations with higher precision and comparable efficiency. The translation errors reported are diverse and distinguished from those found by existing papers [28], [29]. Thus, we believe RTI can compliment with the state-of-the-art approaches. Gupta *et al.* [71] developed a translation testing approach based on pathological invariance: sentences of different meanings should not have identical translation. We did not compare with this paper because it is based on an orthogonal approach and we consider it as a concurrent work.

D. Metamorphic Testing

The key idea of metamorphic testing is to detect violations of metamorphic relations across input-output pairs. Metamorphic testing has been widely employed to test traditional software, such as compilers [72], [73], scientific libraries [74], [75], and service-oriented applications [76], [77]. Because of its effectiveness on testing "non-testable" systems, researchers have also designed metamorphic testing techniques for a variety of AI software. Typical examples include autonomous cars [16], [78], statistical classifiers [79], [80], and search engines [81]. In this paper, we introduce a novel metamorphic testing approach for machine translation software.

VII. Conclusion

We have presented a general concept—referentially transparent input (RTI)—for testing machine translation software. In contrast to existing approaches, which perturb a word in natural sentences (*i.e.*, the context is fixed) and assume that the translation should have only small changes, this paper assumes the RTIs should have similar translations across different contexts. As a result, RTI can report different translation errors (*e.g.*, errors in the translations of phrases) of diverse kinds and

thus complements existing approaches. The distinctive benefits of RTI are its simplicity and wide applicability. We have used it to test Google Translate and Bing Microsoft Translator and found 123 and 142 erroneous translations respectively with the state-of-the-art running time, clearly demonstrating the ability of RTI—offered by *Purity*—to test machine translation software. For future work, we will continue refining the general approach and extend it to other RTI implementations, such as using verb phrases as RTIs or regarding whole sentences as RTIs, pairing them with the concatenation of a semantically-unrelated sentence. We will also launch an extensive effort on translation error diagnosis and automatic repair for machine translation systems.

REFERENCES

[1] B. Zhang, D. Xiong, and J. Su, "Accelerating neural transformer via an average attention network," in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[2] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," in *Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, and I. Kaiser, Lukasz abd Polosukhin, "Attention is all you need," in *Proc. of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[4] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[5] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li *et al.*, "Achieving human parity on automatic chinese to english news translation," *arXiv preprint arXiv:1803.05567*, 2018.

[6] B. Turovsky. (2016) Ten years of google translate. [Online]. Available: https://blog.google/products/translate/ten-years-of-google-translate/

[7] Facebook. (2019) How do i translate a post or comment written in another language? [Online]. Available: https://www.facebook.com/help/509936952489634/

[8] Twitter. (2019) About tweet translation. [Online]. Available: https://help.twitter.com/en/using-twitter/translate-tweets

[9] A. Okrent. (2016) 9 little translation mistakes that caused big problems. [Online]. Available: http://mentalfloss.com/article/48795/9-little-translation-mistakes-caused-big-problems

[10] F. Macdonald. (2015) The greatest mistranslations ever. [Online]. Available: http://www.bbc.com/culture/story/20150202-the-greatest-mistranslations-ever

[11] T. Ong. (2017) Facebook apologizes after wrong translation sees palestinian man arrested for posting 'good morning'. [Online]. Available: https://www.theverge.com/us-world/2017/10/24/16533496/facebook-apology-wrong-translation-palestinian-arrested-post-good-morning

[12] G. Davies. (2017) Palestinian man is arrested by police after posting 'good morning' in arabic on facebook which was wrongly translated as 'attack them'. [Online]. Available: https://www.dailymail.co.uk/news/article-5005489/Good-morning-Facebook-post-leads-arrest-Palestinian.html

[13] T. W. Olympics. (2018) 15,000 eggs delivered to norwegian olympic team after google translate error. [Online]. Available: https://www.nbcwashington.com/news/national-international/Google-Translate-Fail-Norway-Olympic-Team-Gets-15K-Eggs-Delivered-473016573.html

[14] B. Royston. (2018) Israel eurovision winner netta called 'a real cow' by prime minister in auto-translate fail. [Online]. Available: https://metro.co.uk/2018/05/13/israel-eurovision-winner-netta-called-a-real-cow-by-prime-minister-in-auto-translate-fail-7541925/

[15] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proc. of the 26th Symposium on Operating Systems Principles (SOSP)*, 2017.

[16] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *ICSE*, 2018.

[17] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," in *Proc. of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[18] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "Adversarial example generation with syntactically controlled paraphrase networks," in *Proc. of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

[19] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," in *Proc. of the 26th Annual Network and Distributed System Security Symposium (NDSS)*, 2019.

[20] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *Proc. of the 25th USENIX Security Symposium (USENIX Security)*, 2016.

[21] Y. Qin, N. Carlini, I. Goodfellow, G. Cottrell, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *Proc. of the 36th International Conference on Machine Learning (ICML)*, 2019.

[22] J. Ebrahimi, D. Lowd, and D. Dou, "On adversarial examples for character-level neural machine translation," in *Proc. of the 27th International Conference on Computational Linguistics (COLING)*, 2018.

[23] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," in *Proc. of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[24] W. Zheng, W. Wang, D. Liu, C. Zhang, Q. Zeng, Y. Deng, W. Yang, P. He, and T. Xie, "Testing untestable neural machine translation: An industrial case," *arXiv preprint arXiv:1807.02340*, 2018.

[25] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[26] P. K. Mudrakarta, A. Taly, M. Sundararajan, and K. Dhamdhere, "Did the model understand the question?" in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[27] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," in *Proc. of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[28] P. He, C. Meister, and Z. Su, "Structure-invariant testing for machine translation," in *ICSE*, 2020.

[29] Z. Sun, J. M. Zhang, M. Harman, M. Papadakis, and L. Zhang, "Automatic testing and improvement of machine translation," in *ICSE*, 2020.

[30] H. Søndergaard and P. Sestoft, "Referential transparency, definiteness and unfoldability," *Acta Informatica*, 1990.

[31] P.-Y. Saumont. (2017) What is referential transparency? [Online]. Available: https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report

[32] M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu, "Fast and accurate shift-reduce constituent parsing," in *Proc. of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Aug. 2013, pp. 434–443.

[33] Google translate. [Online]. Available: https://translate.google.com

[34] Bing microsoft translator. [Online]. Available: https://www.bing.com/translator

[35] Translation errors for referentially transparent inputs. [Online]. Available: https://github.com/ReferentialTransparency/RTI

[36] S. Kevin. (2018) Why functional programming? the benefits of referential transparency. [Online]. Available: https://sookocheff.com/post/fp/why-functional-programming/

[37] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep., 1998.

[38] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering (TSE)*, vol. 42, 2016.

[39] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 51, 2018.

[40] Y. Liu and M. Sun, "Contrastive unsupervised word alignment with non-local features," in *Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[41] A. Fraser and D. Marcu, "Measuring word alignment quality for statistical machine translation," *Computational Linguistics*, 2007.

[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[43] Glove. [Online]. Available: https://nlp.stanford.edu/projects/glove/

[44] spacy. [Online]. Available: https://spacy.io/

[45] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[46] R. Sennrich, B. Haddow, and A. Birch, "Improving neural machine translation models with monolingual data," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

[47] C. Chu, R. Dabre, and S. Kurohashi, "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)*, 2017.

[48] O. r. Bojar, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, P. Koehn, and C. Monz, "Findings of the 2018 conference on machine translation (wmt18)," in *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, October 2018, pp. 272–307. [Online]. Available: http://www.aclweb.org/anthology/W18-6401

[49] fairseq: A fast, extensible toolkit for sequence modeling. [Online]. Available: https://github.com//pytorch/fairseq

[50] M. Post, "A call for clarity in reporting bleu scores," 2018.

[51] C. Ziegler. (2016) A google self-driving car caused a crash for the first time. [Online]. Available: https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report

[52] F. Lambert. (2016) Understanding the fatal tesla accident on autopilot and the nhtsa probe. [Online]. Available: https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/

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

[54] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proc. of the 35th International Conference on Machine Learning (ICML)*, 2018.

[55] T. Du, S. Ji, J. Li, Q. Gu, T. Wang, and R. Beyah, "Sirenattack: Generating adversarial audio for end-to-end acoustic systems," *arXiv preprint arXiv:1901.07846*, 2019.

[56] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[57] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," in *Proc. of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[58] C. Mao, Z. Zhong, J. Yang, C. Vondrick, and B. Ray, "Metric learning for adversarial robustness," in *Proc. of the 35th Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[59] G. Tao, S. Ma, Y. Liu, and X. Zhang, "Attacks meet interpretability: Attribute-steered detection of adversarial samples," in *Proc. of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[60] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, "Adversarial sample detection for deep neural network through model mutation testing," in *ICSE*, 2019.

[61] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *ASE*, 2018.

[62] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: Automated neural network model debugging via state differential analysis and input selection," in *Proc. of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018.

[63] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *ICSE*, 2019.

[64] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *arXiv preprint arXiv:1906.10742*, 2019.

[65] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: model-based quantitative analysis of stateful deep learning systems," in *Proc. of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019.

[66] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proc. of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2019.

[67] D. Pruthi, B. Dhingra, and Z. C. Lipton, "Combating adversarial misspellings with robust word recognition," in *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[68] M. T. Ribeiro, S. Singh, and C. Guestrin, "Semantically equivalent adversarial rules for debugging nlp models," in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[69] Y. Cheng, Z. Tu, F. Meng, J. Zhai, and Y. Liu, "Towards robust neural machine translation," in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[70] Y. Cheng, L. Jiang, and W. Macherey, "Robust neural machine translation with doubly adversarial inputs," in *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[71] S. Gupta, P. He, C. Meister, and Z. Su, "Machine translation testing via pathological invariance," in *The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020.

[72] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2014.

[73] C. Lidbury, A. Lascu, N. Chong, and A. F. Donaldson, "Many-core compiler fuzzing," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2015.

[74] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *ASE*, 2014.

[75] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels," *Software Testing, Verification and Reliability (STVR)*, vol. 26, no. 3, 2016.

[76] W. K. Chan, S. C. Cheung, and K. R. Leung, "Towards a metamorphic testing methodology for service-oriented software applications," in *Proc. of the 5th International Conference on Quality Software (QSIC)*, 2005.

[77] ——, "A metamorphic testing approach for online testing of service-oriented software applications," *International Journal of Web Services Research (IJWSR)*, vol. 4, no. 2, 2007.

[78] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic autonomous driving system testing," in *ASE*, 2018.

[79] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers," in *Proc. of the 9th International Conference on Quality Software (QSIC)*, 2009.

[80] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software (JSS)*, vol. 84, 2011.

[81] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering (TSE)*, vol. 42, 2016.