Testing Coreference Resolution Systems without Labeled Test Sets

Jialun Cao
The Hong Kong University of Science and Technology,
Guangzhou HKUST Fok Ying Tung Research Institute
Hong Kong, China
jcaoap@cse.ust.hk

Yaojie Lu
Chinese Information Processing Laboratory,
Institute of Software, Chinese Academy of Sciences
Beijing, China
luyaojie@iscas.ac.cn

Ming Wen
School of Cyber Science and Engineering,
Huazhong University of Science and Technology
Wuhan, China
mwenaa@hust.edu.cn

Shing-Chi Cheung*
The Hong Kong University of Science and Technology,
Guangzhou HKUST Fok Ying Tung Research Institute
Hong Kong, China
scc@cse.ust.hk

ABSTRACT

Coreference resolution (CR) is a task to resolve different expressions (*e.g.*, named entities, pronouns) that refer to the same real-world entity/event. It is a core natural language processing (NLP) component that underlies and empowers major downstream NLP applications such as machine translation, chatbots, and question-answering. Despite its broad impact, the problem of testing CR systems has rarely been studied. A major difficulty is the shortage of a labeled dataset for testing. While it is possible to feed arbitrary sentences as test inputs to a CR system, a test oracle that captures their expected test outputs (coreference relations) is hard to define automatically. To address the challenge, we propose CREST, an automated testing methodology for CR systems. CREST uses constituency and dependency relations to construct pairs of test inputs subject to the same coreference. These relations can be leveraged to define the metamorphic relation for metamorphic testing. We compare CREST with five state-of-the-art test generation baselines on two popular CR systems, and apply them to generate tests from 1,000 sentences randomly sampled from CoNLL-2012, a popular dataset for coreference resolution. Experimental results show that CREST outperforms baselines significantly. The issues reported by CREST are all true positives (*i.e.*, 100% precision), compared with 63% to 75% achieved by the baselines.

CCS CONCEPTS

• **Software and its engineering** → **Consistency**; **Software defect analysis**; • **Computing methodologies** → **Information extraction**.

KEYWORDS

Coreference resolution testing, Metamorphic testing, SE4AI

* Corresponding author.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3616258

ACM Reference Format:
Jialun Cao, Yaojie Lu, Ming Wen, and Shing-Chi Cheung*. 2023. Testing Coreference Resolution Systems without Labeled Test Sets. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616258

1 INTRODUCTION

Coreference resolution (CR) is a core natural language processing (NLP) task that resolves a real-world entity/event to which a pronoun or phrase in a text refers [20, 27, 56, 80]. The first example in Figure 2 illustrates an application of CR, resolving the pronoun *him* to *John*, and *she* refers to *Sally*. CR brings cohesion and coherent presentation style to natural language expressions [20], and also permeates many aspects of daily life.

CR is an essential component that empowers many downstream NLP applications. The second to fifth examples in Figure 2 illustrate the use of CR results in four typical NLP applications. Example 2 shows a user talking with her AI assistant Siri, asking it to send a message to *John*. Siri needs to resolve the pronoun *I* to *Sally* and *him* to *John* before correctly sending John a message. The Google Translate in Example 3 needs to resolve the pronoun *it* to *the funeral* before correctly translating the English *it* to French *elle* in the feminine singular form. The question-answering (QA) system in Example 4 needs to resolve the pronoun *she* to *Sally* in the context in order to return the correct answer. A recent study even pointed out that ChatGPT[58] is limited in handling coreference [41]. Indeed, various applications such as question-answering (QA) [48], machine translation [67], chatbots [12, 36, 51, 77], and automatic summarization [4, 42] perform predictions based on the CR results.

Given the wide impact of CR systems, their quality assurance is crucial to the performance of NLP applications. A recent study [80] benchmarked existing CR systems and revealed the potential bias in the reported evaluation of these systems. In particular, the evaluation is based on the commonly-used dataset (*i.e.*, CoNLL-2012 dataset [56]), on which existing CR systems are expected to have been tuned. As such, high precision and recall are reported. The results are believed to be satisfactory in this dataset, while the evaluation results are different when these systems are tested using other datasets or those that involve infrequent tokens. The precision and

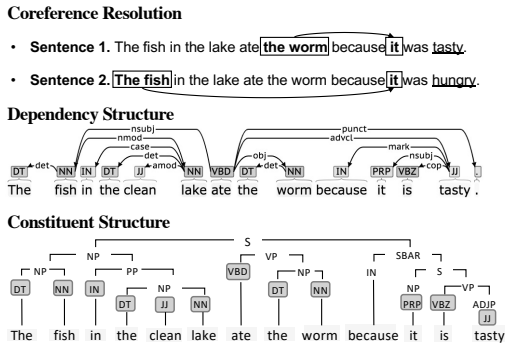


Figure 1: Illustration of Coreference Resolution, Dependency Structure, and Constituent Structure.

recall can drop to 66.5% and 72.4% respectively. The study indicates the need for a diverse test set to evaluate CR systems' reliability.

Indeed, a deficient CR system can greatly jeopardize the performance of its downstream applications [67]. As pointed out by Google, Microsoft, and many other companies, an unreliable CR system has been one of the major reasons for the unsatisfactory performance of downstream applications such as knowledge graph construction [33] and neural machine translation [67]. For instance, Example 5 in Figure 2 shows a dialog where a chatbot failed to resolve the pronoun *him* to *John*. It repeatedly sought clarification about the message's receiver, resulting in a poor user experience.

Metamorphic testing [11] is reported to be the most popular testing approach for artificial intelligence systems [45]. Metamorphic testing relieves (1) the construction of a test oracle between an input and its output, which is difficult for stochastic computation, and (2) a labeled test set, which is expensive to collect. No prior work has studied the deployment of automated metamorphic testing to CR systems despite their impact. Suppose $\mathcal{F}$ is the program under test;$\mathcal{F}(x)$ and $\mathcal{F}(x')$ are the program outputs based on an input x and a follow-up input x'. The deployment of metamorphic testing needs to address three key problems: (**P1**) defining an effective metamorphic relation (MR) $\mathcal{R}(x, x', \mathcal{F}(x), \mathcal{F}(x'))$ for CR systems, (**P2**) generating a follow-up input x' that observes the defined MR from a source input x and its output $\mathcal{F}(x)$, and (**P3**) devising a mechanism to check if $\mathcal{F}(x)$ and $\mathcal{F}(x')$ violate $\mathcal{R}$ automatically.

P1. To address the first problem, a natural MR is that the CR system should produce consistent results for a source input and its follow-up input when both share the same coreference.

P2. The MR requires constructing a follow-up input that preserves the coreference of its source input. The construction is, however, challenging. First, existing word/token replacement techniques [22, 24, 47, 64] do not guarantee the preservation of coreference after replacement. Indeed, the coreference in a sentence could be easily changed with the replacement of one word. For example, in Figure 1, with one adjacent word (*i.e.*, *hungry* and *tasty*), the coreference of *it* is changed from *the fish* to *the worm*. Also, apart from word replacement, adversarial techniques could also be used for data augmentation (typo insertion or random character deletion) [47, 64]. Yet, the design of these techniques cannot guarantee the coreference that should be preserved. Second, coreference may also be changed by replacing a word with its synonyms/antonyms. For instance, consider the sentence "*The city councilmen refused the demonstrators a permit because they [feared/advocated] violence.*"[75] If the word is "*feared*", then "*they*" likely refers to "*The city councilmen*"; whereas if it is "*advocated*", then "*they*" likely refers to "*the demonstrators*". Third, as revealed by our study (see Section 4), existing semantic similarity metrics [16, 62] are incapable of distinguishing whether two sentences have the same coreference.

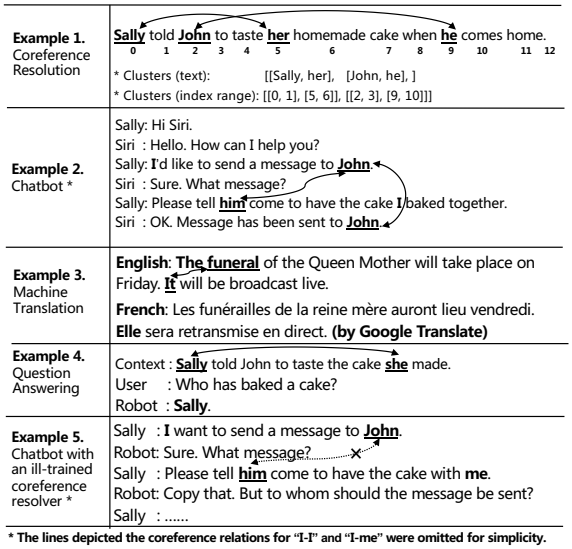


Figure 2: Examples of Coreference Resolution and Its Downstream Applications.

P3. Validating the consistency of two results made by CR systems can be tricky. Intuitively, directly checking text consistency could lead to ambiguity [49]. For example, considering the sentence:

- *The Queen Mother asked Queen Elizabeth II*$_{[1]}$ *to transform* ***her***$_{[1]}$ *sister, Princess Margaret*$_{[2]}$*, into a viable princess by summoning a renowned speech therapist, Nancy Logue, to treat* ***her***$_{[2]}$ *speech impediment.* [49]

In this example, there are two pronouns *her* referring to different referents (*i.e.*, *Queen Elizabeth II* and *Princess Margaret*, respectively). So simply determining the consistency of texts could lead to false negatives. Alternatively, if we use the clusters of index range (as shown in Figure 2 Example 1) to determine the consistency, the index could easily shift due to word replacement. For example, if we replace the word *homemade* with its synonym *home-baked*, the index range of *he* would shift from `[9,10]` to `[11,12]`, because the synonym *home-baked* will be tokenized to three tokens, so the indices of *home-baked* are `[6]`, `[7]`, `[8]`, respectively, pushing the index of *him* forward to `[11]`. In other words, simply checking the absolute equivalence of either text or index range could result in imprecise judgements.

In this paper, we introduce a metamorphic testing framework, Crest, for CR systems. Our key insight is that replacing tokens that are related to the coreference in the text is likely to change the coreference of the text. Thus, accordingly, we generate the follow-up inputs by mutating the phrases or tokens that are unrelated to the coreference in the text. In particular, we identify the CR-related tokens/phrases according to the syntactic information (*i.e.*, constituent structure and dependency structure) of the text, then generate the follow-up inputs by replacing tokens in the rest of the

source input. For example, for Sentence 1 in Figure 1, after identifying the coreference in the text (*i.e.*, *the worm* and *it*) and parsing the text to its dependency and constituent structures, we can identify that not only the tokens/phrases *the worm* and *it*, but also the adjective *tasty*, which depends on the preceding nominal subject (`nsubj`) *it* from the dependency structure, are CR-related tokens/phrases. Replacing these tokens/phrases is highly likely to affect the text's coreference. So, we leave these tokens/phrases unchanged. Instead, we replace the rest tokens/phrases in the source input. Specific to this example, it is safe to replace the tokens in the phrase *in the clean lake* without affecting the coreference. After the word replacement, we also check the consistency of the constituency of the CR-related tokens/phrases, ensuring that the word replacement would not change the coreference accidentally. Finally, to validate the consistency of the outputs, we check the clusters of index range and the corresponding clusters of texts at the same time. Also, to better quantify the output consistency, we calculate the precision and recall using a popular-used link-based metric BLANC (BiLateral Assessment of Noun-phrase Coreference) [61], so that the desired thresholds can be customized for finer consistency requirements. In summary, our work makes three major contributions:

- We introduce Crest, a metamorphic testing methodology for CR systems. In particular, we formulate the testing problem on CR systems, demonstrate the challenges of testing them, and develop a feasible methodology to address the challenges.
- We propose a metamorphic relation for testing CR systems. It addresses the difficulty in automatically constructing test inputs during generating follow-up inputs in aid of syntax analysis. A subsequent test selection process further ensures the quality of test inputs.
- The experiment reveals the effectiveness of Crest. It reaches 100% precision in issue detection, outperforming baselines (with at best 74%). Moreover, regarding the quality of test inputs, only 63% ∼ 79% of test inputs generated by baselines are valid, leading to high false positive rates. In comparison, 100% test inputs generated by Crest are valid and are able to reveal true positives.

2 PRELIMINARIES

2.1 Technology and Terminology of CR

CR is a fundamental task and a long-researched area in the NLP area towards natural language understanding [27, 56]. To automate CR tasks, a variety of techniques have been proposed, including deterministic [5, 17, 38, 50, 59, 60], statistical [13, 23] and neural [14, 15] CR systems. Besides, recent end-to-end techniques [39, 40] have made a big step by making use of the pre-trained language model (PLM) [34], external world knowledge [3] and efficient learning algorithms [35, 78]. Yet, as a recent study [80] has pointed out that the performance of state-of-the-art CR systems is still unsatisfactory when infrequent test sets are used.

A desired CR system is expected to identify all the expressions (*e.g.*, names, pronouns, phrases) that co-referring to an ***entity/event*** in the given text [6, 27, 49]. These expressions are known as ***mentions***. The mentions are grouped into ***clusters***[1] according to the entity/events they refer to. All the mentions in one cluster are expected to refer to the same entity/event. For instance, in the text "*The fish in the lake ate the worm because it was tasty.*", a CR system is expected to find all mentions {*The fish*, *the lake*, *the worm*, *it*} from the text. Next, the CR system partitions them into singletons {*the fish*} and {*the lake*}, and a cluster {*the worm*, *it*}, and then outputs the three clusters.[2]

2.2 Output Cluster Consistency Checking

Given a text sequence $\mathcal{X} = [x_1, x_2, \ldots, x_N]$, a ***mention*** $m = (s, e)$, where s and e are, respectively, the start and end token position of m such that $s \leq e \leq N$. Let c denote a ***cluster*** $\{m_1, m_2, \ldots, m_j\}$, where $j \geq 2$. Given a CR system $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{C}$, it takes a text $\mathcal{X}$ as input, and outputs a set of clusters $\mathcal{C}$ (=$\{c_1, \ldots, c_q\}$), where $q \geq 1$. Take Example 1 in Figure 2 as an example, *mentions* include `[2,3]` (*i.e. John*) and `[9,10]` (*i.e. he*), the cluster of these two mentions is `[[2,3], [9,10]]`.

For clusters c_1 and c_2, we say c_1 and c_2 are ***consistent*** if $\forall m \in c_1, \exists m' \in c_2$ such that $m \equiv m'$ AND $\forall m' \in c_2, \exists m \in c_1$ such that $m' \equiv m$. Note that the equivalence relation (*i.e.*, $\equiv$) of mentions describes the equivalence beyond the exact index equivalence. For instance, for the example in Figure 2 Example 1 , although the index of *he* is shifted from `[9]` to `[11]` after replacing *homemade* to *homebaked*, these two mentions are still equivalent because they both use the same pronoun (*i.e.*, *he* to refer to the same entity *John*).

To evaluate the coreference consistency, we then adopt BLANC (BiLateral Assessment of Noun-phrase Coreference) [43, 61], a popular-used metric, to calculate the precision and recall of the coreference. Formally, suppose $\mathcal{G}$ is the ground truth coreference of the given input $\mathcal{X}$ consisting of a set of clusters, $\mathcal{C}$ is the predicted coreference of $\mathcal{X}$. Let L_g and L_c represent the sets of coreference links in the ground truth clusters $\mathcal{G}$ and predicted clusters $\mathcal{C}$.

$$P = |L_g \cap L_c| \;/\; |L_c|, \qquad R = |L_g \cap L_c| \;/\; |L_g| \tag{1}$$

We consider $\mathcal{C}$ is ***correct*** if both P and R are higher than the customized thresholds. Oppositely, if any of P and R lowers than the threshold, an issue is detected.

2.3 Constituent and Dependency Structures

Constituent and dependency structures are two ways to analyze the syntactic structure of sentences using constituency [85] and dependency parsing [9] respectively. The constituent structure represents each sentence as a hierarchical phrase tree, where each node represents a constituent [71], such as noun phrases (NP), verb phrases (VP), and prepositional phrases (PP). For example, in Figure 1, the main constituents of the sentence are {*The fish in the clean lake*, *ate the worm*, *because it is tasty*}. Specifically, the noun phrase (NP) *'The fish'* is the subject of the sentence, and the prepositional phrase (PP) *'in the clean lake'* modifies the noun phrase *'The fish'*, *'ate the worm'* is the verb phrase (VP) of the sentence with *'ate'* as the verb and *'the worm'* as the direct object, and *'because it is tasty'* is the subordinate clause (SBAR) that explains the reason for the fish eating the worm.

The dependency structure represents a sentence as a directed graph with the words as nodes and dependency types as edges [52].

[1] There are other alternative names for *clusters*, including *coreference chains* and *subjects*. In this paper, we use *cluster* consistently.

[2] Most CR systems do not output singletons. So in the evaluation, we assume each output cluster of the CR system contains at least two mentions.

Each dependency represents the relationship between a headword and its dependents, which are words that depend on the headword for their meaning. As shown in Figure 1, *fish* depends on the verb *ate*, *worm* is the direct object of the verb *ate* and depends on it, *clean* modifies *lake* and depends on it, *because* introduces a subordinate clause and depends on *ate*, *tasty* is an adjective that modifies *worm* and depends on it.

3 APPROACH AND IMPLEMENTATION

This section introduces the workflow of Crest and describes its implementation. The input of Crest is a list of unlabeled texts, and the output is a list of suspicious CR issues. For each input, Crest generates a set of follow-up inputs, pairs each follow-up input with its source input, and reports an issue if any inconsistency is found. A reported issue contains a source input, a follow-up input, and the coreference of them resolved by the CR system under test. Note that the issue may occur in the coreference of the source input, the coreference of the follow-up input, or both. Figure 3 illustrates the overview of Crest. setCrest comprises three main steps:

- **Follow-up Input Generation**: For each source input, Crest generates a set of follow-up inputs by modifying a single token at a time while preserving the coreference.
- **Follow-up Input Selection**: Crest further selects the generated texts according to the syntactic information of the source and follow-up inputs.
- **Inconsistency Detection**: The predicted coreference of the source and follow-up inputs are compared. If the predictions are inconsistent, Crest reports an issue.

3.1 Follow-up Input Generation

Crest designs the methodology following a general workflow of metamorphic testing. To start with, it generates a set of follow-up inputs from a source input according to the MR. The MR in this work is that a CR system should produce consistent results for the source and follow-up inputs when they share the same coreference. The follow-up input can be so constructed that it differs from the source input by at least one token. However, most token replacement mechanisms in NLP testing approaches [7, 22, 24, 30, 69] do not consider if there are coreference changes in the follow-up inputs. The follow-up inputs constructed in this way may not satisfy the immutability requirement of the coreference in the MR. Though several approaches consider coreference [64, 66], they manually define several templates (*e.g.*, `[He/She]` is a `[doctor]`) to generate tests. While these approaches are limited in diversifying the generated sentence structures, it is impractical to define templates for all sentence structures.

It motivates us to design a generation algorithm (Algorithm 1), which observes the immutability requirement of coreference as follows. The input of the algorithm includes a source input $\mathcal{X}$ (*e.g.*, a sentence), the golden coreference C in the dataset, the CR system under test $\mathcal{F}$, and the maximum number of generated follow-up inputs M. Given a source input $\mathcal{X}$ (*e.g.*, a sentence) and the golden coreference C, at the beginning, the source input is tokenized and parsed to the dependency tree and constituent tree (lines 2-3), obtaining the constituent labels and the dependency relations of tokens. Examples of parsed constituent structure and dependency structure are illustrated in Figure 1. Then, according to the identified coreference and the dependency structure, we obtain the index of tokens that are related to the coreference (line 4, see next paragraph for details). Then, each token $\mathcal{X}$ looks up its replacement iteratively unless the token is coreference-related (lines 6-7). For each candidate token, we find the corresponding synonym and antonym sets (lines 10-11) using WordNet [46]. Furthermore, in order to increase the diversity of the generated follow-up inputs, we also involve off-the-shelf masked language models to perform word perturbation. In particular, we mask the corresponding token in the original sentence one by one and use a pre-trained language model to predict a possible token as a replacement (line 12)[3]. After obtaining a set of words that can be replaced, we then try to replace it in order to generate a set of follow-up inputs (line 13).

Specifically, to obtain the index of coreference-related tokens (Algorithm 1, line 4), we utilize dependency structures of texts. Given a source input $\mathcal{X}$, its coreference C and dependency structure *dept*, it iteratively obtains clusters in C, and breaks down to each mention in every cluster. For every mention, we add all the indices of the mention, as well as that of the phrases/tokens that have dependency relations with these indices. The formal equation for obtaining the set of CR-related indices ($\mathbb{N}$) is shown as follows:

$$\mathbb{N} = \{j \mid \exists\, i \in \mathbb{M}, \forall j \in \{0, 1, \ldots |X|\},\ hasDependency(i, j)\} \quad (2)$$

where j iterates all the indices in the sentence X with |X| tokens, $\mathbb{M}$ denotes the set of indices of tokens in the coreference, i denotes any index in $\mathbb{M}$, $hasDependency(i, j)$ represents a function which checks whether there are dependency relations[4] between the indices i and j, which returns a boolean value. We add the index j into the returned set $\mathbb{N}$ if there are dependency relations with i and j. Specifically, for the dependency relations, we consider `nsubj` (*i.e.*, nominal subject, a nominal which is the syntactic subject and the proto-agent of a clause [52]) and `amod` (*i.e.*, adjectival modifier, the adjectival phrase/word that serves to modify the noun/pronoun [52]) of pronouns in the sentence, because they are two major dependency relations that relate to noun/noun phrases. Finally, we return a set of indices $\mathbb{N}$.

Furthermore, after a token that could be used for replacement is identified, the word replacement should go with a double check utilizing the constituent structure. In particular, the procedure works as follows (see Algorithm 3). Given a set of candidate tokens *replacedSet* (*i.e.*, synonyms or antonyms), the source input $\mathcal{X}$, the index of token under replacement i, the constituent structure of $\mathcal{X}$ *const*, and the maximum replacement times of the given position (*i.e.*, i) M , the algorithm deals with every candidate token in the *replacedSet* one by one. After replacing the i-th token in the source text with every candidate token (line 6), it parses the new text to its constituent structure (line 7) to check whether the candidate text has the same constituent label as the i-th token in the source text (line 8). If the label remains the same, then the new text is regarded as a follow-up input and added into the output set. This

[3] In the implementation, we use a popular model (*i.e.*, *bert-large-cased*) to perform word perturbation and to ensure the naturalness of the sentence, we set the minimal probability of each candidate word as 0.1. One could easily replace the language model with more advanced models and set up different thresholds. Since it is not the core of our methodology, we follow the same setting as adopted by the existing work [69].
[4] In the implementation of Crest, we use the dependency parser provided by StanfordCoreNLP [44].

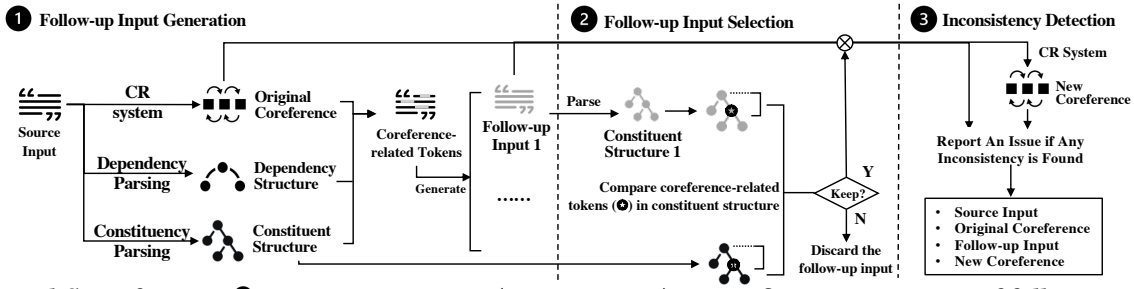


Figure 3: Workflow of CREST. ❶ Given a source input (*e.g.*, a sentence), CREST first generates a set of follow-up inputs (*e.g.*, sentences that are similar to the source input while differing in a few tokens) in the first phase. ❷ The follow-up inputs are fed to the second phase, validating their syntactic property consistency with the source input. A follow-up input if consistent is paired with its concerned source input to form a test; otherwise discarded. ❸ An issue is detected and reported if the consistency of the output coreference is below a predefined threshold.

Algorithm 1: generateFollowUpInput

Input: A source input $\mathcal{X}$, the golden coreference C of $\mathcal{X}$, CR system under test $\mathcal{F}$, the number of maximum generated follow-up inputs M

Output: A set of generated follow-up inputs *newXs*

```
1  newXs ← emptySet()
2  dept ← dependencyParser(X)
3  const ← constituentParser(X)
4  crind ← getCrfRelatedIndex(C, dept)
5  for i in 0 ... |X| do
6      if i in crind then
7          continue
8      else
9          candidates ← emptySet()
10         candidates.add (getSynonym(X, i))
11         candidates.add (getAntonym(X, i)
12         candidates.add (getWordPerturb(X, i)
13         newXs.add(replaceAndCheck(candidates, X, i,
             const, M))  ◃ See Algorithm 3
14 return newXs
```

check ensures the replacement does not change the constituency of the token under replacement. The replace-and-check iterates until the number of generated follow-up inputs reaches the maximum number M (**lines** 3-4) or all tokens in the candidate set have been replaced (line 10).

3.2 Follow-up Input Selection

We further analyze the generated follow-up inputs and exclude those that are not likely to preserve their source input's coreference. Yet, checking whether coreference has been changed is as difficult as asserting whether it is correct. Checking the coreference manually is, however, not a viable solution because it can be subjective and not scalable. Our insight to tackle this problem is inspired by rule-based CR works [38, 59, 60], which leverage the depth of the token/phrase in a syntactic tree to determine discourse prominence [28] and resolve coreference. Therefore, we check if the coreference-related tokens in a follow-up input's constituent structure have a depth that differs from that of their counterparts in the source input's structure. If so, the follow-up input is unlikely to preserve its source input's coreference and should be excluded.

In particular, a mention in a cluster could be either a single token (*e.g.*, *him*, *it*), or a phrase (*e.g.*, *the fish*, *Executives at Backer Spielvogel client Avis Inc.*). So when calculating the depth of coreference-related tokens/phrases, we treat each mention accordingly. If a mention is a single token, we calculate the depth from the root to this token in the constituent structure. While if a mention is a phrase, we calculate the depth from the root to the closet nested phrase outside the mention phrase. For example (see Figure 1 Constituent Structure), for the single token *it*, the depth is 5 (`S - SBAR - S - NP - PRP`), while for the mention *the fish*, its depth is calculated from the root to the nested noun phrase (`S - NP - NP`), resulting in 3.

Algorithm 2 works as follows. Given a source input $\mathcal{X}$ and a follow-up input $\mathcal{X}'$, the algorithm determines whether the follow-up input should be excluded or not. If the coreference-related tokens reside differently in two constituency structures, then the follow-up input should be excluded. Specifically, the algorithm iterates every mention in clusters of the coreference in $\mathcal{X}$ (line 1-2). If a mention appearing in $\mathcal{X}$ also exists in the follow-up input, then CREST parses $\mathcal{X}'$ to obtain its constituent structure (line 3-4). Then it calculates the depth of the token or the closet phrase that nests the mention in the constituent structures (*const* and *const'*) of both $\mathcal{X}$ and $\mathcal{X}'$ (line 5-6). If the depth varies, then we discard the follow-up input (line 7-9). Otherwise, if no mentions break the condition till the end of the iteration, we then keep this follow-up input (line 9).

3.3 Inconsistency Detection

After the follow-up inputs are selected, each of them are paired with its source input. For each pair of source input and follow-up input $(\mathcal{X}, \mathcal{X}')$, a test is conducted by (1) obtaining the coreference of $\mathcal{X}$ resolved by the CR system under test $\mathcal{F}(\mathcal{X})$, (2) obtaining the coreference of $\mathcal{X}'$ resolved by the CR system under test $\mathcal{F}(\mathcal{X}')$, and (3) checking if $\mathcal{F}(\mathcal{X})$ and $\mathcal{F}(\mathcal{X}')$ are consistent. The test fails if inconsistency occurs.

Yet, validating the coreference consistency of two outputs by a CR system is tricky because there can be more than one equivalent token/phrase in a sentence. Thus, we use the token indices to represent the mentions in coreference to avoid ambiguity. For instance, for the example in Section 1 P3, instead of using the text *her* to represent the mention, we use `[9,10]` to represent the first *her* referring to *Queen Elizabeth II*, and use `[31,32]` to represent

Algorithm 2: selectFollowUpInput

Input: A source input $\mathcal{X}$, a generated follow-up input $\mathcal{X}'$, the constituent structure *const*, the coreference C
Output: A boolean value indicating whether the generated follow-up input should be kept or not

```
for cluster in C do
    for mention in cluster do
        if mention in X' then
            const' ← ConstituentParser(X')
            depth ← calDepth(mention, const)
            depth' ← calDepth(mention, const')
            if depth != depth' then
                return False
return True;
```

Algorithm 3: replaceAndCheck

Input: A source input $\mathcal{X}$, the index of the token under replacement i, a set of candidate tokens that could be used for word replacement *replacedSet*, the constituent structure *const*, and the maximum number of follow-up inputs M
Output: A set of follow-up inputs *newXs*

```
newXs ← emptySet()
for newT in replacedSet do
    if len(newXs ≤ M) then
        return newXs
    else
        newX ← replaceToken(X, i, newT)
        newConst ← constituentParser(newX)
        if newConst(newT) == const(X[i]) then
            newXs.add(newX)
return newXs;
```

the second *her* referring to *Princess Margaret*. The use of indices allows each mention to be uniquely referenced. .

Finally, CREST detects inconsistency as follows. Given a follow-up input $\mathcal{X}'$, CREST feeds $\mathcal{X}'$ to the CR system to obtain the output coreference C'. To avoid ambiguity of text comparing, CREST uses token indices to represent mentions in coreference. Then, an inconsistency is detected if C' differs from C. CREST determines inconsistency according to their computed BLANC (BiLateral Assessment of Noun-phrase Coreference) [61] score, a widely-used link-based metric for coreference measurement. If the BLANC score is less than 1.00, an inconsistency is detected. CREST reports the source $\mathcal{X}$ and follow-up inputs $\mathcal{X}'$ of the detected inconsistency, together with their coreference C and C'.

4 EVALUATION

Four research questions (RQs) are designed to evaluate CREST:

- **RQ1. Are issues reported by CREST true positives?** CREST detects issues of CR systems when the metamorphic relation is violated. We investigate how likely an issue reported by CREST is a true positive. We show the number of tests generated, detected issues, and the true positive rates of each approach. We also report the overhead and generalizability of CREST.
- **RQ2. Can follow-up inputs generated by CREST preserve the coreference well?** Since the metamorphic relation in CREST assumes the generated follow-up inputs preserve the coreference as the source inputs, we thus check whether this assumption holds well. We compare the number of follow-up inputs that preserved or not preserved coreference for each approach, and point out that the high false positive rates could largely be attributed to the coreference-changed follow-up inputs.
- **RQ3. What kinds of coreference issues can CREST reveal?** We analyze the issues found by CREST and categorized them into six types. We explain each issue type with examples reported by CREST and illustrate the statistics of the reported issues.
- **RQ4. What is the impact of each step in the CREST?** We conduct experiments using different setups and thresholds to show the impact of each step in the designed method.

4.1 Evaluation Setup

We implement CREST in Python, and conduct experiments on a MacBook Pro with 2.3 GHz Quad-Core Intel Core i7 CPU, 2.3 GHz, 16 GB memory. For each source input, we set the maximum number of generated follow-up inputs for each given source input to 20. One could enlarge this number to find more inconsistencies. Moreover, we set the threshold for both precision and recall of BLANC (*i.e.*, thr_p and thr_r) to 1.0.

Dataset. We use CoNLL (the Conference on Natural Language Learning)-2012 [56] for evaluation because it is the most widely used evaluation benchmark for coreference resolution [80].[5] This dataset could also be used for other natural language processing tasks such as semantic role labeling, name entity recognition, and so on. We use the English annotation in the dataset for evaluation. Specifically, CoNLL-2012 provides the training set (22,965 sentences), the validation set (2,791 sentences), and the test set (2,894 sentences) separately. In evaluation, source inputs are randomly selected from the test set. In addition, in this paper, we refer to the ground truth provided by the dataset annotated following the OntoNotes [55] or determine the correct coreference following the same annotation standard.

Baselines. For comparison, since there is no available testing approach for CR system, we adapt three recent testing approaches, SIT [24], PatInv [22] and CAT [69], for other NLP tasks (*i.e.*, neural machine translation) as baselines. These approaches are selected because they also generate texts by replacing a few tokens of the original text, and pairing the generated and the original texts to compare their output results. In particular, SIT [24] generates new texts using BERT by replacing one noun and adjective token with its synonyms. CAT [69] considers the context when replacing the synonyms, and replaces tokens regardless of the part of speech. PatInv [22], on the other hand, replaces tokens with non-synonyms, aiming to diversify the generated texts. Besides, for behavioral correctness and robustness, CheckList [64] and TextAttack generate

[5] There are intra- and cross-sentence coreference annotations. The difference is whether the mentions of coreference cross the sentence or not. In our evaluation, we use the intra-sentence coreference data to conduct the evaluation for simplicity, while our methodology is applicable to both types of coreference.

texts by transformation such as typo insertion (*e.g.*, replacing `his` with `hi s` by adding a space) and changing named entities. We also include them as baselines. In particular, for CheckList, we use all the transformation methods provided except *negation*. This is because the metamorphic relation may no longer hold after negating the original text's semantic meaning. For TextAttack, which bundles various adversarial attack and data augmentation techniques, we exclude those methods that have been used in CheckList to avoid duplication, including random character deletion and augmentation techniques based on word embedding. For comparison, we run these baselines based on their artifacts using their default settings.

CR System Under Test. We evaluate the effectiveness of Crest on two CR systems. One is a neural-based CR system proposed by Clark and Manning [14, 15]. For implementation, we use the NeuralCoref implementation provided by spaCy [29][6]. The other is a mention-ranking statistical CR system [13], using the implementation provided by Stanford's CoreNLP library [44][7]. For both systems, we use the default configurations.

4.2 RQ1. Effectiveness of Issue Revealing

To evaluate the effectiveness of Crest, we randomly sample 1,000 sentences from the test set of CoNLL-2012, and adopt various approaches to generate follow-up inputs based on these source inputs. For each source input, we set an upper bound of 20 to limit the number of generated follow-up inputs for fairness. Among the 1,000 source inputs, 535 of them are correctly resolved using spaCy [29] with the NeuralCoref algorithm, while 465 of them are not.

The number of generated follow-up inputs (# Gen), the number of detected issues (# Issue) are shown in Table 1. We can see that the number of issues generated and detected by Crest almost doubles than those by most of the baselines. In particular, the baselines find thousands of issues (488 ~ 7,348). Although TextAttack reports the most issues (7,348), nearly 40% of them are false alarms. Two factors contribute to the inferior performance of the baselines. First, some baselines restrict the candidate replacement tokens to specific part-of-speech tags, and such a restriction limits the number of test input candidates that can be generated. For example, SIT considers nouns and adjectives for replacement, while PatInv mainly considers verbs and adverbs for replacement. Second, some baselines adopt a strict test input selection criterion. For example, CAT requires high syntactic and semantic similarity between the source and generated sentences, thus excluding many sentences that are valid test inputs. Note that it is true that adjusting the parameters in baselines may probably yield more follow-up inputs while exploring better parameters of baselines is not our main goal, thus we leave it for future exploration.

For the recall evaluation, it is impractical to go through all the source and follow-up inputs. So, we check the hits based on source inputs, *i.e.*, whether the 465 wrongly-resolved source inputs could be reported by any of the pairs that these inputs involve. The result is shown in Table 1 "# (%) Hits". It shows that the baselines recall 230 to 463 of the wrongly-resolved source inputs, with an average of 349. Crest hits 324 wrongly-resolved source inputs, which approaches the average. Besides, it is noteworthy that TextAttack achieves a hit rate of 99.57%. Yet, a high hit rate could be trivially achieved by raising alarms for every test. Therefore, we further evaluate the baselines and Crest by examining the number of true positives and false positives of the detected issues to assess its effectiveness in terms of precision. In particular, a true positive means that the pair of sentences reported by the approach has at least one error in the predicted coreference. A false negative means the reported pair of sentences turn out to have no coreference errors.

Table 1: Effectiveness of Issue Revealing on NeuralCoref

	Test Generation					Error Detection		
	# Src (P+N)	# Gen	# Issues	# (%) Hits	Time	# TP	# FP	Prec
SIT	1,000 (535 + 465)	10,488	2,338	230 / 465 (49.46)	0.02	1,652	686	0.71
PatInv	1,000 (535 + 465)	9,445	1,881	167 / 465 (35.91)	0.44	1,398	483	0.74
CAT	1,000 (535 + 465)	16,640	2,197	425 / 465 (91.4)	0.96	1,531	666	0.73
Checklist	1,000 (535 + 465)	3,744	488	334 / 465 (71.83)	**0.01**	350	138	0.72
TextAttack	1,000 (535 + 465)	20,000	7,348	**463 / 465 (99.57)**	0.27	**4,648**	2,656	0.64
Crest	1,000 (535 + 465)	13,635	1,457	324 / 465 (69.68)	0.21	1,457	**0**	**1.00**

Table 1 "Error Detection" shows that the precision (Prec) of baselines ranges from 64% to 74% as compared to 100% achieved by Crest. It is clear that the false positive rates of baselines are high. Especially, the high hit rate achieved by TextAttack is at the cost of low precision (64%). Essentially, the unsatisfactory results are attributed to the change of the coreference during follow-up input generation, making the inconsistency a false alarm.

We further elaborate on the detected issues using three examples. First, we use an issue on NeuralCoref revealed by Crest after replacing it with synonyms. In Example 6, after replacing *first* with its synonym *initiative*, no coreference in the follow-up input such constructed can be identified. Crest reports it as an issue.

Example 6. An issue found by Crest by synonym replacement. (Replace: *first* → *initiative*)

∨ *Just before Christmas break,* ***the rural township of*** ***Linpien, on the Pingtung Plain at the mouth of the*** ***Linpien river***[1], *held* ***its***[1] <***first***> *ever Wax Apple Festival.*

× *Just before Christmas break, the rural township of Linpien, on the Pingtung Plain at the mouth of the Linpien river,*[1 *is missed*] *held its*[1 *is missed*] <***initiative***> *ever Wax Apple Festival.*

Alternatively, if we replace a coreference-unrelated token with its antonym, the coreference should also preserve. As shown in Example 7, after replacing *different* with an antonym *same*, the coreference in the text is still preserved. However, the CR system under test fails to identify the second cluster in the text. Crest reports an issue accordingly.

Example 7. An issue found by Crest by antonym replacement. (Replace: *different* → *same*)

∨ ***We***[1] *have heard a* <***different***> *sign of Sir Paul McCartney playing to enthusiastic crowds in Red Square, and* ***we***[1] *have been enthralled by the celebrations of* ***your own city***[2] *in* ***Petersburg***[2].

× ***We***[1] *have heard a* <***same***> *sign of Sir Paul McCartney playing to enthusiastic crowds in Red Square, and* ***we***[1] *have been enthralled by the celebrations of your own city*[2 *is missed*] *in Petersburg*[2 *is missed*].

The third example is a false positive reported by a baseline. In Example 8, without considering the coreference during word replacement, SIT substitutes *heart* for *pair*, which is a mention in the coreference cluster. In that case, the CR system fails to identify any coreference from the second sentence, thus an inconsistency is

[6] spaCy Nerualcoref: https://spacy.io/universe/project/neuralcoref
[7] Stanford CoreNLP: https://stanfordnlp.github.io/CoreNLP/coref.html

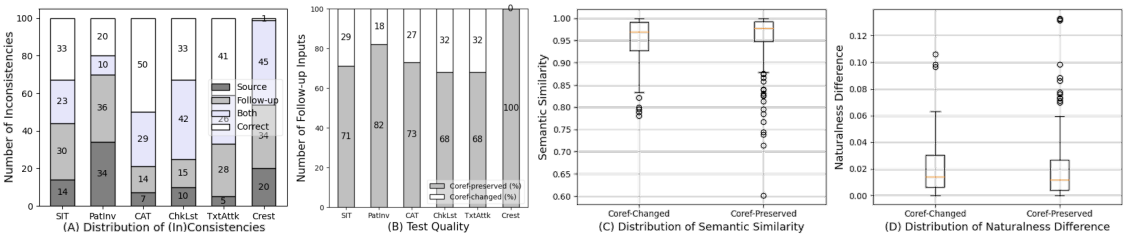


Figure 4: Analysis on Test Quality and Distribution of Inconsistencies.

reported. Yet, it is actually a false alarm, because the substitution changes the coreference, not only the co-referring entity, but also the multiplicity, where *the heart* in singular form cannot be referred to by *them* unlike *the pair*.

Example 8. A false alarm (Replace: *pair* → *heart*)

∨ *She asks anyone who finds* ***the*** *<**pair**>*$_{[1]}$ *to give* ***them***$_{[1]}$ *back.*
∨ *She asks anyone who finds the <**heart**> to give them back.*

In addition, an issue could be induced by the incorrect coreference of the source input, the follow-up input, or both, so we further categorize the inconsistencies reported by each approach on NeuralCoref according to where the issues occur. The analysis is conducted on the 100 pairs sampled from each baseline. Figure 4 (A) shows the distribution. The black bars (*Source*) show the number of inconsistencies in which the coreference of the source inputs is wrongly resolved, while the coreference of the follow-up input is correct. Likewise, the gray bars (*Follow-up*) means the number of inconsistencies where the coreference of the follow-up input goes wrong. The silver bars (*Both*) mean both are wrong. While the white bars (*False Alarm*) show the number of false positives. Figure 4 (A) shows that the *Source* contributes less than the *Follow-up*, meaning that more issues are revealed only by the follow-up inputs than by the source inputs in the evaluation. PatInv generates the most issue-revealing follow-up inputs (36), followed by our work (34). Also, apart from our work, the number of false positives reported by baselines is high. We analyze the reason in Section 4.3.

Overhead. We present the overhead of generating follow-up inputs for each approach. As shown in Table 1 (Time), the average time to generate a follow-up input ranges from 0.01 to 0.96 seconds. On the one hand, an approach could be quick if it only finds and replaces words using pattern matching or rules. On the other hand, the overhead comes mainly from the syntactic and semantic analysis. For example, it takes CAT around 1 second to generate one follow-up input because CAT filters them by checking whether the substituted tokens are semantically similar to the original tokens. Similarly, Crest takes around 0.27 second to generate a follow-up input because it requires syntactic and coreference analysis.

Generalizability. To investigate if the results can be generalized to other CR systems, we repeat the previous experiments of Crest on a popular statistical CR system [13]. The results are shown in Table 2. Among 1,000 source sentences, 635 of them can be correctly resolved by statistical CR, while 365 cannot. The number of generated follow-up inputs is similar to that in Table 1. Regarding the number of hits, 21.37% to 99.45% incorrectly resolved source inputs are included in the inconsistencies detected. Besides, Crest outperforms baselines significantly in precision, reaching

Table 2: Effectiveness of Issue Revealing on Statistic CR

	Test Generation					Error Detection		
	# Src (P+N)	# Gen	# Issues	# (%) Hits	Time	# TP	# FP	Prec
SIT	1,000 (635+365)	10,488	2,174	143 / 365 (39.18)	0.07	1,422	752	0.65
PatInv	1,000 (635+365)	9,445	1,410	78 / 365 (21.37)	0.51	1,059	351	0.75
CAT	1,000 (635+365)	16,640	2,513	324 / 365 (88.77)	1.09	1,697	816	0.68
Checklist	1,000 (635+365)	3,744	479	199 / 365 (54.52)	0.03	312	167	0.65
TextAttack	1,000 (635+365)	20,000	7,470	**363 / 365 (99.45)**	0.36	4,596	2,874	0.62
Crest	1,000 (635+365)	13,625	1,357	232 / 365 (63.56)	0.46	1,357	**0**	**1.00**

100% precision, compared with 75% at best reached by baselines. The experiment shows the generalizability of Crest in high precision in issue revealing on both neural and statistical CR systems.

To explore why Crest misses some wrongly resolved source sentences, we increase the upper bound of follow-up inputs to 50, but no more hits are found (see Table 3). We find two reasons for this situation. First, the CR system is incapable to resolve or tends to ignore certain tokens (*e.g.*, rare words) or structures (*e.g.*, Sentences with noun phrases with long attributives). For such defects, it is potential to use these sentences as templates for training data augmentation in order to fine-tune the CR system. Second, the span of output coreference differs from that in the ground truth. For example, one mention in a predicted coreference is *power suppliers* while the ground truth is *power suppliers in the west* . The differences in output spans may arise from the different boundary annotations that CR systems adopt [84]. For such defects, one may either stick to an annotation standard or relax the criteria on inconsistency detection (*e.g.*, lower the thresholds for recall or precision).

4.3 RQ2. Quality of Follow-up Inputs

We further analyze the quality of follow-up inputs by comparing the number of follow-up inputs that preserve the coreference as the source input. In particular, since the results on the two CR systems are similar, in the following, we illustrate the analysis on NeuralCoref due to space limitation. The full experiment results can be found online [2].

The result is shown in Figure 4 (B). The gray bars represent the number of follow-up inputs that preserve their source inputs' coreference, while the white bars mean the opposite. We can see that the baselines only preserve 68% to 82% among all the generated follow-up inputs. In comparison, 100% of the follow-up inputs generated by Crest preserve the coreference, indicating the high fidelity of the test inputs generated by Crest.

Although some baselines check certain constraints such as semantic similarity within the given contexts [69] and syntactic invariants [24], they do not consider the preservation of coreference in test input generation. PatInv [22] aims to generate dissimilar text, which could be effective when the generated text preserves the coreference. Baselines using adversarial attacks, however, could

easily change coreference in the text. For example, transformations like random character deletion (*e.g.*, perturbing *they* to *the*) and typo insertion (*e.g.*, changing *his* to *their country* to *heir country* change coreference in the text, leading to invalid follow-up inputs.

We further analyze whether coreference changes in a text are correlated to the changes in its semantics or naturalness. The analysis helps us to answer whether existing approaches designed for the similarity of semantics or preservation of naturalness could be deployed to generate effective test inputs for CR systems. Note that in order to better measure the semantic similarity at a finer granularity, we adopt the latest tool [30] to capture the subtle semantic difference between two texts. In particular, for naturalness measurement, we adopt the following equation [30]:

$$Naturalness(X) = \sqrt[|X|]{\prod_{i=1}^{|X|} P(X_i|X_{\setminus i})} \quad (3)$$

which measure the naturalness of sentence X in length of $|X|$ tokens. X_i represents the i-th token in the sentence X, and $X_{\setminus i}$ represents the tokens except the i-th token.

The statistics are shown in Figure 4 (C). The semantic similarity ranges similarly regardless of whether the coreference is changed. In particular, the average similarity between coreference-changed pairs is 0.972, while that between coreference-preserved pairs is 0.969, which is quite close to 0.972, with only 0.003 difference. Also, the similarity between coreference-changed pairs is a bit higher than the counterpart, indicating that a high semantic similarity does not necessarily lead to a high possibility to preserve the coreference. Thus, it is impractical to distinguish whether the coreference has been changed using semantic similarity.

Similarly, we also show the change in naturalness to investigate whether the change in language fluency could provide hints to detect coreference changes. As shown in Figure 4 (D), the ranges of coreference-changed (0.000 to 0.106) and -preserved (0.00 to 0.133) are quite close, averaging at 0.0316 and 0.0313, respectively. Such subtle differences in naturalness could not be used to distinguish whether the coreference is changed. Our analysis suggests that test generation techniques based on coreference are needed.

Human Evaluation. To better evaluate the soundness of Crest, we conducted a human study on the generated follow-up sentences. We published the task on Prolific [57], a crowd-sourcing platform that connects researchers with study participants for online research studies. We recruited five participants and set the nations of participants within the USA, UK, Australia, Canada, and Singapore in order to ensure the participants are native speakers. We set five participants following existing patterns [6]. In addition, we also invited two experienced and professional English communication tutors. In particular, one is an experienced trainer in Linguistics and Methodology, and the other has over 30-year experience in teaching English. To select the sentences to be labeled, we randomly sampled 100 pairs of original and follow-up sentences from the 13,689 pairs (in Table 1) generated by Crest for labeling.[8]

The labeling results are shown in Figure 5 (B). Each bar represents the statistics from one participant. `CT1` and `CT2` refer to two English communication tutors, `User1-5` refer to five participants hired from Prolific. The result (the dashed line in Figure 4 (B)) shows that more than half (66.7%) of the source sentences are considered more natural than their follow-up counterparts, which is reasonable because the source sentences are taken from newsletters, Wikipedia, and other reading resources that professional writers authored. The results also show that 33% of the follow-up sentences are considered equally or more natural than the original sentences. Indeed, 7.7% of the follow-up sentences are considered to be more natural than the source sentences. Further analysis of the result reveals that only 21% source sentences are considered consistently by all the participants to be more natural than their follow-ups. In other words, most follow-up sentences (79%) are considered natural or equally natural by at least one participant. The study demonstrates the naturalness of the generated follow-up sentences, thus indicating the soundness of our methodology.

[8]To avoid repeatedly selecting the same original input sentence, We first sample 100 original sentences, then randomly select one follow-up from all the follow-ups for each original sentence.

4.4 RQ3. Issues Reported by Crest

Crest is capable to find CR issues of diverse types. In our evaluation, Crest finds six major types of CR issues from NeuralCoref [14], including Span Error (SE), Missing Entity (ME), Extra Entity (EE), Missing Mention (MM), Extra Mention (EM), and Conflated Entities (CE). The issue types are derived from the error types of CR systems [37]. To ease the understanding, we provide the corresponding examples of the uncovered issues in each type.

4.4.1 ***Extra Entity (EE).*** If a CR system identifies an extra entity (*i.e.*, cluster) in the given text, it is an Extra Entity issue. The example below shows that a CR system identifies additional co-referring cluster *the arts* and *the art*. Though they look similar, they are referring to different abstract concepts.

Example 9. An Example of Extra Entity (EE)

× *The congressman*$_{[1]}$ *was known as a fervent liberal and supporter of* ***the arts***$_{[2\times]}$ *and* *he*$_{[1]}$ *was noted for* *his*$_{[1]}$ *success in getting congress to finance for national endowment for* ***the art***$_{[2\times]}$.

4.4.2 ***Missing Entity (ME).*** Opposite to *Extra Entity*, *Missing Entity* describes an issue where the entity should have been identified yet is missed. Examples are omitted due to space limitation.

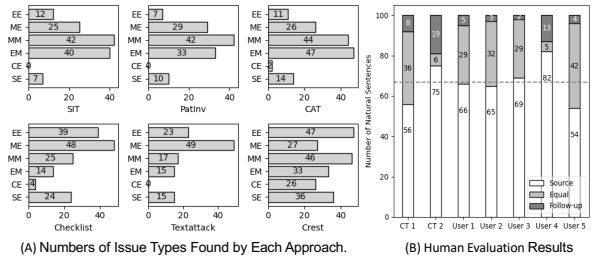
4.4.3 ***Conflated Entity (CE).*** If a CR system mistakenly merges two or more clusters of entities into one, a CE issue occurs. See Example 10, there are three clusters of entities in the sentence, while in the output clusters, the CR system mistakenly merges the second and the third clusters together, causing a CE issue.

Example 10. An Example of Conflated Entity (CE)

∨ *The Scotts*$_{[1]}$ *ran it past* *their*$_{[1]}$ *vet*$_{[2]}$ *and* *the vet*$_{[2]}$ *assured* *them*$_{[1]}$*, as long as* *the hen*$_{[3]}$ *left* *her*$_{[3]}$ *crate to do business,* *she*$_{[3]}$ *was no hurt to the pup.*

× *The Scotts*$_{[1]}$ *ran it past* *their*$_{[1]}$ *vet*$_{[2]}$ *and* *the vet*$_{[2]}$ *assured* *them*$_{[1]}$*, as long as* *the hen*$_{[2\times]}$ *left* ***her***$_{[2\times]}$ *crate to do business,* ***she***$_{[2\times]}$ *was no hurt to the pup.*

4.4.4 ***Extra Mention (EM).*** If a cluster contains one or more mentions that are not co-referring to the entity as the remaining mentions in that cluster, it is regarded as an Extra Mention issue. For example, in the following sentence, the name *Tanshui* and



(A) Numbers of Issue Types Found by Each Approach. (B) Human Evaluation Results

Figure 5: Bar Charts of Error Type Distribution (A) and Naturalness of Generated Sentences (B).

its pronoun *her* are correctly identified, while *the giant beast of development* is unrelated to *Tanshui*, let alone referring to it.

Example 11. An Example of Extra Mention (EM)

× *Lovers of* $\underline{Tanshui}_{[1]}$*, while keeping watch over* $\underline{her}_{[1]}$*, are fearful that* $\underline{\textbf{\textit{the giant beast of development}}}_{[1\times]}$ *may eventually come and bulldoze* $\underline{Tanshui}_{[1]}$ *away one piece of land at a time.*

4.4.5 ***Missing Mention (MM).*** Opposite to EM, if any mention in any cluster is missing, then it is a Missing Mention issue. MM issues usually happen in long sentences where there are more than two mentions referring to one entity/event.

4.4.6 ***Span Error (SE).*** A SE issue describes the situation where the span of the identified mention is larger or smaller than the ground-truth mention. For example, in Example 12, the ground truth CR is {*such a fever chip, it*}, while the CR system identifies the wrong span for the first mention as follows.

Example 12. An Example of Span Error (SE)

∨ *However, Pan points out that the usefulness of* $\underline{\textbf{\textit{such a fever chip}}}_{[1]}$ *is still limited so that* $\underline{it}_{[1]}$ *cannot yet be widely promoted.*

× *However, Pan points out that* $\underline{\textbf{\textit{the useful of of such a fever chip}}}_{[1\times \textit{Span Error}]}$ *is still limited, so that* $\underline{it}_{[1\vee]}$ *cannot yet be widely promoted.*

Furthermore, we show the statistics of issue types found by each approach in Figure 5 (A). Note that an incorrect coreference could be attributed to more than one issue type, and the coreference of both source and follow-up inputs could be wrong. From the figure, we can see that Crest could reveal issues in diverse types, and the number of each issue type keeps a better balance than baselines. On the contrary, baselines tend to reveal more Missing Mention (MM) and Extra Mention (EM) issues than other issue types and are ineffective in revealing Conflated Entity (CE) issues.

Besides, Divided Entity [37] could not be revealed by any of the approaches. The reason is that this issue type usually occurs in document-level coreference (*i.e.*, the coreference across sentences), while we use the intra-sentence coreference (*i.e.*, the coreference occurs within a sentence) for evaluation.

4.5 RQ4. Impact of Each Step

There are three main steps in Crest. To explain the impact of each step, we conduct the following experiments.

4.5.1 ***Impact of Step 1 (Follow-up Input Generation).*** The first step of Crest is to generate follow-up inputs of a given source input. In this step, the parameter M prescribes the maximum number of follow-up inputs for one source input. To quantify the importance of step 1, we set M as 10, 20, 30, and 50, respectively, to show the impact on the effectiveness. The experiment settings and the CR system under test are the same as that in Table 1. The experiment is shown in Table 3. As M increases (the entry 'M'), the number of follow-up inputs increases accordingly, ranging from 7,947 to 21,503, and more inconsistencies ('# Issues') are reported, from 1,313 to 2,070. Besides, note that the hit rate stays stable, meaning that with at most 10 follow-ups generated, around 85% incorrect coreference in the source inputs can be successfully identified, while after that, simply increasing the number of generated follow-ups can hardly hit more faults in the source coreference. Regarding the precision, we can see that with step 2 (*i.e.*, follow-up input selection) enabled, the precision of Crest stays at 100% regardless of how many issues are reported. Therefore, we conclude that the impact of Step 1 mainly lies in the number of revealed issues, and would not affect the precision of Crest.

Table 3: Impact of Steps 1 and 2 of Crest. The configuration M means the number of maximum generated follow-up inputs (in Algorithm 1) in Step 1 of Crest. The configuration 'Filter' denotes whether Step 2 of Crest is conducted (*w/*) or not (*w/o*). The values that are sensitive to the configuration are highlighted in yellow.

Configuration		Test Generation			Error Detection			Coref-Preserving (%)
M	Select	# Gen	# Issues	# (%) Hits	# TP	# FP	Prec (%)	
10	w/	7,947	1,314	324 / 465 (69.68)	1,314	**0**	**100.00**	**7,947 / 7,947 (100.00)**
20	w/	13,635	1,457	324 / 465 (69.68)	1,457	**0**	**100.00**	**13,635 / 13,635 (100.00)**
30	w/	16,611	1,488	324 / 465 (69.68)	1,488	**0**	**100.00**	**16,611 / 16,611 (100.00)**
50	w/	18,698	1,519	324 / 465 (69.68)	1,519	**0**	**100.00**	**18,698 / 18,698 (100.00)**
10	w/o	9,112	1,755	397 / 465 (85.38)	1,611	**144**	**91.79**	7,947 / 9,122 (87.12)
20	w/o	15,630	1,965	397 / 465 (85.38)	1,796	**169**	**91.40**	13,635 / 15,630 (87.24)
30	w/o	19,047	2,021	397 / 465 (85.38)	1,841	**180**	**91.09**	16,611 / 19,047 (87.21)
50	w/o	21,503	2,071	397 / 465 (85.38)	1,878	**193**	**90.68**	18698 / 21503 (86.96)

4.5.2 ***Impact of Step 2 (Follow-up Input Selection).*** Step 2 further filters out the follow-up inputs according to the CR-related tokens in the grammar structure. In particular, we show the results when step 2 is enabled or disabled. The result is shown in Table 3 (the entry 'Select'). We can see that when step 2 is disabled (*i.e.*, *w/o*), the precision drops to around 91% compared with 100% when it is enabled (*i.e.*, *w/*), and the ratio of coreference preserved follow-ups drops from 100% to around 87%. Also, when step 2 is disabled, the number of true positives decreases by 297 (= 1611 - 1314) ~ 359 (= 1878 - 1518), accounting for 0.18 (=297/1611) ~ 0.19 (= 359/1878) percent decrease. To sum up, the result validates the significance of step 2 from two aspects. First, by filtering out about 13% coreference-changed follow-ups, step 2 makes sure the precision of error detection reaches 100%, a 10% increase from that without the selection. Second, step 2 keeps a high precision with slightly fewer true positives detected, and without hit rate decreases.

4.5.3 ***Impact of Step 3 (Inconsistency Detection).*** In step 3, Crest reports an inconsistency if any of the precision and recall of the coreference is lower than 1.00 (*i.e.*, $thr_p = thr_r = 1.0$). We use two examples to show their impacts. Example 13 shows that when thr_p drops while the thr_r remains to be 1.00, there are extra coreference links are mistakenly identified from the sentence. In particular, there is only one link in the ground-truth (∨), while three links are identified, so the precision of coreference is 0.33 (= 1/3). While for the recall of coreference, since all the coreference links in the ground truth are identified, the recall is 1.0. Example 14 shows

that when thr_r drops while the thr_p reserves, there are coreference links that should be identified yet missed. Example 14 shows the situation. In the ground truth, there are seven coreference links, while only six of them are recalled, so the recall of coreference is 0.86 (= 6/7), while the precision stays at 1.0.

Example 13.

```
∨ [[‘the city’, ‘this city’]]
× [[’this city’, ’it’, ’the city’]]
```

Example 14.

```
∨ [[‘himself’,‘God’,‘his’,‘he’], [‘his people’,‘They’]]
× [[‘his’,‘God’,‘he’,‘himself’]]
```

To sum up, a higher thr_p requires more correct coreference links regardless of the number of missing links, while thr_r requires more links to be identified though there may be incorrect links involved.

5 RELATED WORK

5.1 Testing Techniques for NLP Systems

Various testing methodologies have been proposed for specific NLP tasks such as neural machine translation [8, 10, 24, 25, 53, 64, 83], code analysis [1, 31, 54], and QA [8]. A recent survey [45] points out that metamorphic testing [11] is the most popular testing approach for AI systems. A recent study [30] benchmarks state-of-the-art metamorphic testing approaches and finds that a large proportion of their generated inputs violate the metamorphic relations, leading to high false positive rates. There are a few recent works that could be used for CR system testing. CheckList [64] lists a matrix of linguistic capabilities (*e.g.*, robustness) and test types (*e.g.*, invariance test). In this work, whether the coreference can be correctly resolved is considered a linguistic capability to be tested in upperstream applications such as machine comprehension. Tests are then generated either by templates or by textual transformations (*e.g.*, inserting typos). Yet, text transformations have high probabilities to change coreference in the sentence. TextFlint [74] provides a multilingual robustness evaluation platform that incorporates universal text transformation, adversarial attack, etc. For CR systems, transformations such as random sentence deletion in a paragraph are proposed. These transformations are only suitable for coreference across sentences (paragraph-level coreference), while our work is suitable for both sentence- and paragraph-level coreference. Another testing technique, ASTRAEA [66] targets the fairness of NLP tasks such as CR . It generates sentences by changing occupations and male/female pronouns to test the fairness of the NLP system under test. Our work complements existing works. It focuses on effectively generating test inputs that respect coreference , which has not been explored by prior techniques.

5.2 Adversarial Techniques for NLP Systems

Adversarial attack techniques aim to produce adversarial examples [70] to confuse AI systems. They perturb benign inputs by applying certain adversarial perturbations, so that the generated adversarial examples successfully confuse the AI system making wrong predictions. In particular, there are various adversarial attack approaches targeting NLP systems, including character-level [19, 26], word-level [21, 81], sentence-level [32, 63], and multiple-level [18, 72] attacks. Several studies summarize the state-of-the-art adversarial attack algorithms [73, 82] and defending algorithms [76]. Besides, several off-the-shelf tool-kits [47, 65, 79] are available to generate adversarial attacks on general [47, 79] or certain [65] NLP tasks. Popular perturbations such as word replacement have been incorporated in the baselines in our evaluation. An adversarial attack technique [8], which assumes all input data to be labeled, was recently proposed for CR systems. This technique is subsumed by the baseline SIT [24], with only the word in the mention replaced with its semantic-related substitutions (*e.g.*, synonym).

6 THREATS OF VALIDITY

We discuss three threats in this work. First, the experimental procedures involve random sampling, which may induce randomness in the results. To mitigate this threat, we sampled 1,000 samples as seeds, which is five times the sample size of prior work [7, 24, 68, 69]. Also, the experimental results show that our work outperforms the baselines significantly. The degree of outperformance surpasses randomness. Two, the number of tests generated by the baselines may be sensitive to parameter settings. To accommodate the variation, experiments are designed to evaluate the quality of generated tests rather than the number of tests generated. In the current evaluation, we adopt the default parameter settings (*e.g.*, parameters, language models) of the baselines to facilitate comparison. Third, the human evaluation of sentence naturalness is conducted over a crowd-sourcing platform, where the education level of the participants varies. To mitigate the threat, we involve two professional English tutors as participants as well.

7 CONCLUSION

In this paper, we introduce Crest, a general methodology for validating CR systems. It tackles the difficulty in automatically constructing test inputs by considering the coreference during test generation in aid of syntax analysis. The experiment reveals the effectiveness of Crest by achieving 100% precision in inconsistency detection, and 100% of the generated tests are of high quality.

8 DATA AVAILABILITY

We release the reproduction package at the website [2] for validation. The package includes (1) source code of Crest, (2) experimental results of comparisons with baselines, and (3) labeling results of human evaluation.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation of China (Grant No. 61932021, 62141210, 62002125), the Hong Kong Research Grant Council/General Research Fund (Grant No. 16205722), and the Hong Kong Research Grant Council/Research Impact Fund (Grant No. R5034-18), the Hong Kong Innovation and Technology Fund (Grant No. MHP/055/19), the Young Elite Scientists Sponsorship Program by CAST (Grant No. 2021QNRC001). The authors would also like to thank the anonymous reviewers for their comments and suggestions and express our appreciation to the participants for conducting the human evaluation.

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* 3, POPL (2019), 1–29.

[2] Anonymous. 2023. CREST. https://github.com/ArabelaTso/Crest-artifacts

[3] Rahul Aralikatte, Heather Lent, Ana Valeria Gonzalez, Daniel Herschcovich, Chen Qiu, Anders Sandholm, Michael Ringaard, and Anders Søgaard. 2019. Rewarding Coreference Resolvers for Being Consistent with World Knowledge. In *Proceedings of the 2019 Conference on EMNLP-IJCNLP*. Association for Computational Linguistics, Hong Kong, China, 1229–1235.

[4] Saliha Azzam, Kevin Humphreys, and Robert Gaizauskas. 1999. Using Coreference Chains for Text Summarization. In *Proceedings of the Workshop on Coreference and Its Applications (CorefApp '99)*. Association for Computational Linguistics, College Park, Maryland, 77–84.

[5] Eric Bengtson and Dan Roth. 2008. Understanding the Value of Features for Coreference Resolution. In *Proceedings of EMNLP*. Association for Computational Linguistics, Honolulu, Hawaii, 294–303.

[6] Ari Bornstein, Arie Cattan, and Ido Dagan. 2020. CoRefi: A Crowd Sourcing Suite for Coreference Annotation. In *Proceedings of EMNLP: System Demonstrations*. Association for Computational Linguistics, Online, 205–215.

[7] Jialun Cao, Meiziniu Li, Yeting Li, Ming Wen, Shing-Chi Cheung, and Haiming Chen. 2022. SemMT: a semantic-based testing approach for machine translation systems. *ACM TOSEM* 31, 2 (2022), 1–36.

[8] Haixia Chai, Wei Zhao, Steffen Eger, and Michael Strube. 2020. Evaluation of Coreference Resolution Systems Under Adversarial Attacks. In *Proceedings of the First Workshop on Computational Approaches to Discourse*. Association for Computational Linguistics, Online, 154–159.

[9] Danqi Chen and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of EMNLP*. Association for Computational Linguistics, Doha, Qatar, 740–750.

[10] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Validation on Machine Reading Comprehension Software without Annotated Labels: A Property-Based Method. In *Proceedings of the 29th ACM Joint Meeting on ESEC/FSE (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 590–602.

[11] Tsong Yueh Chen, S. C. Cheung, and Siu-Ming Yiu. 2020. Metamorphic Testing: A New Approach for Generating Next Test Cases, In Technical Report HKUST-CS98-01. *CoRR* abs/2002.12543, 11. arXiv:2002.12543

[12] Yu-Hsin Chen and Jinho D. Choi. 2016. Character Identification on Multiparty Conversation: Identifying Mentions of Characters in TV Shows. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, Los Angeles, 90–100.

[13] Kevin Clark and Christopher D. Manning. 2015. Entity-Centric Coreference Resolution with Model Stacking. In *Proceedings of ACL and the 7th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, Beijing, China, 1405–1415.

[14] Kevin Clark and Christopher D. Manning. 2016. Deep Reinforcement Learning for Mention-Ranking Coreference Models. In *Proceedings of EMNLP*. Association for Computational Linguistics, Austin, Texas, 2256–2262.

[15] Kevin Clark and Christopher D. Manning. 2016. Improving Coreference Resolution by Learning Entity-Level Distributed Representations. In *Proceedings of ACL*. Association for Computational Linguistics, Berlin, Germany, 643–653.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the NAACL: Human Language Technologies, NAACL-HLT*. Association for Computational Linguistics, Online, 4171–4186.

[17] Greg Durrett and Dan Klein. 2013. Easy Victories and Uphill Battles in Coreference Resolution. In *Proceedings of the 2013 Conference on EMNLP*. Association for Computational Linguistics, Seattle, Washington, USA, 1971–1982.

[18] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of ACL*. Association for Computational Linguistics, Melbourne, Australia, 31–36.

[19] Steffen Eger, Gözde Gül Şahin, Andreas Rücklé, Ji-Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, and Iryna Gurevych. 2019. Text Processing Like Humans Do: Visually Attacking and Shielding NLP Systems. In *Proceedings of the 2019 Conference of the NAACL: Human Language Technologies*. ACL, Minneapolis, Minnesota, 1634–1647.

[20] Pradheep Elango. 2005. Coreference resolution: A survey. *University of Wisconsin, Madison, WI* 1, 12 (2005), 12.

[21] Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking NLI Systems with Sentences that Require Simple Lexical Inferences. In *Proceedings of ACL*. Association for Computational Linguistics, Melbourne, Australia, 650–655.

[22] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. 2020. Machine Translation Testing via Pathological Invariance. In *Proceedings of the 28th ACM Joint Meeting on ESEC/FSE (ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 863–875.

[23] Aria Haghighi and Dan Klein. 2010. Coreference Resolution in a Modular, Entity-Centered Model. In *Proceedings. of NAACL*. Association for Computational Linguistics, USA, 385–393.

[24] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-Invariant Testing for Machine Translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 961–973.

[25] Pinjia He, Clara Meister, and Zhendong Su. 2021. Testing Machine Translation via Referential Transparency. In *43rd IEEE/ACM International Conference on Software Engineering, Madrid, Spain*. IEEE, Madrid, Spain, 410–422.

[26] Xuanli He, Lingjuan Lyu, Lichao Sun, and Qiongkai Xu. 2021. Model Extraction and Adversarial Transferability, Your BERT is Vulnerable!. In *Proceedings of the 2021 Conference of the NAACL: Human Language Technologies*. Association for Computational Linguistics, Online, 2006–2012.

[27] Lynette Hirschman and Nancy Chinchor. 1998. Appendix F: MUC-7 Coreference Task Definition (version 3.0). In *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*. Proceedings of a Conference Held in Fairfax, Virginia, Fairfax, Virginia, 17.

[28] J Hobbs. 1986. *Resolving Pronoun References*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 339–352.

[29] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

[30] Jen-tse Huang, Jianping Zhang, Wenxuan Wang, Pinjia He, Yuxin Su, and Michael R. Lyu. 2022. AEON: A Method for Automatic Evaluation of NLP Test Cases. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 202–214.

[31] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of ACL*. Association for Computational Linguistics, Berlin, Germany, 2073–2083.

[32] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial Example Generation with Syntactically Controlled Paraphrase Networks. In *Proceedings of the 2018 Conference of the NAACL: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 1875–1885.

[33] Heng Ji and Joel Nothman. 2016. Overview of TAC-KBP2016 Tri-lingual EDL and Its Impact on End-to-End KBP. In *Proceedings of the 2016 Text Analysis Conference, TAC 2016, Gaithersburg, Maryland, USA, November 14-15, 2016*. NIST, USA, 15.

[34] Mandar Joshi, Omer Levy, Luke Zettlemoyer, and Daniel Weld. 2019. BERT for Coreference Resolution: Baselines and Analysis. In *Proceedings of the 2019 Conference on EMNLP-IJCNLP*. Association for Computational Linguistics, Hong Kong, China, 5803–5808.

[35] Yuval Kirstain, Ori Ram, and Omer Levy. 2021. Coreference Resolution without Span Representations. In *Proceedings of the 59th Annual Meeting of the ACL and the 11th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, Online, 14–19.

[36] Satwik Kottur, José M. F. Moura, Devi Parikh, Dhruv Batra, and Marcus Rohrbach. 2018. Visual Coreference Resolution in Visual Dialog Using Neural Module Networks. In *Proceedings of ECCV*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 160–178.

[37] Jonathan K. Kummerfeld and Dan Klein. 2013. Error-Driven Analysis of Challenges in Coreference Resolution. In *Proceedings of the 2013 Conference on EMNLP*. Association for Computational Linguistics, Seattle, Washington, USA, 265–277.

[38] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. Stanford's Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Portland, Oregon, USA, 28–34.

[39] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end Neural Coreference Resolution. In *Proceedings of the 2017 Conference on EMNLP*. Association for Computational Linguistics, Copenhagen, Denmark, 188–197.

[40] Kenton Lee, Luheng He, and Luke Zettlemoyer. 2018. Higher-Order Coreference Resolution with Coarse-to-Fine Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 687–692.

[41] Xianzhi Li, Xiaodan Zhu, Zhiqiang Ma, Xiaomo Liu, and Sameena Shah. 2023. Are ChatGPT and GPT-4 General-Purpose Solvers for Financial Text Analytics? An Examination on Several Typical Tasks. arXiv:2305.05862 [cs.CL]

[42] Zhengyuan Liu, Ke Shi, and Nancy F. Chen. 2021. Coreference-Aware Dialogue Summarization. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue, SIGdial 2021, Singapore and Online, July 29-31, 2021*, Haizhou Li, Gina-Anne Levow, Zhou Yu, Chitralekha Gupta, Berrak Sisman, Siqi Cai, David Vandyke, Nina Dethlefs, Yan Wu, and Junyi Jessy Li (Eds.). Association for Computational Linguistics, Singapore and Online, 509–519.

[43] Xiaoqiang Luo, Sameer Pradhan, Marta Recasens, and Eduard Hovy. 2014. An Extension of BLANC to System Mentions. In *Proceedings of ACL*. Association for Computational Linguistics, Baltimore, Maryland, 24–29.

[44] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing

Toolkit. In *Proceedings of 52nd Annual Meeting of the ACL: System Demonstrations*. Association for Computational Linguistics, Baltimore, Maryland, 55–60.
[45] Silverio Mart'inez-Fern'andez, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software Engineering for AI-Based Systems: A Survey. *ACM Transactions on Software Engineering and Methodology* 31 (2022), 1 – 59.
[46] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (nov 1995), 39–41.
[47] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. In *Proceedings of the 2020 Conference on EMNLP: System Demonstrations*. ACL, Online, 119–126.
[48] Thomas S. Morton. 1999. Using Coreference for Question Answering. In *Proceedings of the Workshop on Coreference and Its Applications (CorefApp '99)*. Association for Computational Linguistics, USA, 85–89.
[49] Vincent Ng. 2017. Machine Learning for Entity Coreference Resolution: A Retrospective Look at Two Decades of Research. In *AAAI (AAAI'17)*. AAAI Press, San Francisco, California, USA, 4877–4884.
[50] Vincent Ng and Claire Cardie. 2002. Improving Machine Learning Approaches to Coreference Resolution. In *Proceedings of the 40th Annual Meeting of the ACL*. ACL, Philadelphia, Pennsylvania, USA, 104–111.
[51] Yulei Niu, Hanwang Zhang, Manli Zhang, Jianhong Zhang, Zhiwu Lu, and Ji-Rong Wen. 2019. Recursive Visual Attention in Visual Dialog. In *IEEE Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, Long Beach, CA, USA, 6679–6688.
[52] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*. European Language Resources Association, Portorož, Slovenia, 1659–1666.
[53] Daniel Pesu, Zhi Quan Zhou, Jingfeng Zhen, and Dave Towey. 2018. A Monte Carlo Method for Metamorphic Testing of Machine Translation Services. In *3rd IEEE/ACM International Workshop on Metamorphic Testing MET*. ACM, Gothenburg, Sweden, 38–45.
[54] Michael Pradel and Koushik Sen. 2018. Deepbugs: A learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–25.
[55] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards Robust Linguistic Analysis using OntoNotes. In *Proceedings of Computational Natural Language Learning*. Association for Computational Linguistics, Seattle, Washington, USA, 143–152.
[56] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL (CoNLL '12)*. Association for Computational Linguistics, USA, 1–40.
[57] Prolific. 2014. Prolific. https://www.prolific.co/
[58] Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is ChatGPT a General-Purpose Natural Language Processing Task Solver? arXiv:2302.06476 [cs.CL]
[59] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. 2010. A Multi-Pass Sieve for Coreference Resolution. In *Proceedings of the 2010 Conference on EMNLP*. Association for Computational Linguistics, Cambridge, MA, 492–501.
[60] Marta Recasens, Marie-Catherine de Marneffe, and Christopher Potts. 2013. The Life and Death of Discourse Entities: Identifying Singleton Mentions. In *Proceedings of the 2013 Conference of the NAACL: Human Language Technologies*. Association for Computational Linguistics, Atlanta, Georgia, 627–633.
[61] M. Recasens and E. Hovy. 2011. Blanc: Implementing the Rand Index for Coreference Evaluation. *Nat. Lang. Eng.* 17, 4 (oct 2011), 485–510.
[62] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, Online, 3980–3990.
[63] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically Equivalent Adversarial Rules for Debugging NLP models. In *Proceedings of ACL*. Association for Computational Linguistics, Melbourne, Australia, 856–865.
[64] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of ACL*. Association for Computational Linguistics, Online, 4902–4912.
[65] Walter Simoncini and Gerasimos Spanakis. 2021. SeqAttack: On adversarial attacks for named entity recognition. In *Proceedings of the 2021 Conference on EMNLP: System Demonstrations*. Association for Computational Linguistics, Online, 308–318.
[66] Ezekiel Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. 2022. Astraea: Grammar-Based Fairness Testing. *IEEE Transactions on Software Engineering* 48, 12 (2022), 5188–5211.
[67] Dario Stojanovski and Alexander Fraser. 2018. Coreference and Coherence in Neural Machine Translation: A Study Using Oracle Experiments. In *Proceedings of the Third Conference on Machine Translation*. Association for Computational Linguistics, Brussels, Belgium, 49–60.
[68] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic Testing and Improvement of Machine Translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 974–985.
[69] Zeyu Sun, Jie M. Zhang, Yingfei Xiong, Mark Harman, Mike Papadakis, and Lu Zhang. 2022. Improving Machine Translation Systems via Isotopic Replacement. In *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1181–1192.
[70] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*. International Conference on Learning Representations, Banff, Canada, 10.
[71] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. *The Penn Treebank: An Overview*. Springer Netherlands, Dordrecht, 5–22.
[72] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal Adversarial Triggers for Attacking and Analyzing NLP. In *Proceedings of the 2019 Conference on EMNLP-IJCNLP*. Association for Computational Linguistics, Hong Kong, China, 2153–2162.
[73] Wenqi Wang, Run Wang, Lina Wang, Zhibo Wang, and Aoshuang Ye. 2023. Towards a Robust Deep Neural Network Against Adversarial Texts: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 3 (2023), 3159–3179.
[74] Xiao Wang, Qin Liu, Tao Gui, Qi Zhang, Yicheng Zou, Xin Zhou, Jiacheng Ye, Yongxin Zhang, Rui Zheng, Zexiong Pang, Qinzhuo Wu, Zhengyan Li, Chong Zhang, Ruotian Ma, Zichu Fei, Ruijian Cai, Jun Zhao, Xingwu Hu, Zhiheng Yan, Yiding Tan, Yuan Hu, Qiyuan Bian, Zhihua Liu, Shan Qin, Bolin Zhu, Xiaoyu Xing, Jinlan Fu, Yue Zhang, Minlong Peng, Xiaoqing Zheng, Yaqian Zhou, Zhongyu Wei, Xipeng Qiu, and Xuanjing Huang. 2021. TextFlint: Unified Multilingual Robustness Evaluation Toolkit for Natural Language Processing. In *Proceedings of the 59th Annual Meeting of the ACL and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 347–355.
[75] Terry Winograd. 1972. Understanding natural language. *Cognitive Psychology* 3, 1 (1972), 1–191.
[76] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K Jain. 2020. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing* 17, 2 (2020), 151–178.
[77] Xintong Yu, Hongming Zhang, Yangqiu Song, Changshui Zhang, Kun Xu, and Dong Yu. 2021. Exophoric Pronoun Resolution in Dialogues with Topic Regularization. In *Proceedings of EMNLP*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 3832–3845.
[78] Michelle Yuan, Patrick Xia, Chandler May, Benjamin Van Durme, and Jordan Boyd-Graber. 2022. Adapting Coreference Resolution Models through Active Learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 7533–7549. https://doi.org/10.18653/v1/2022.acl-long.519
[79] Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Zixian Ma, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2021. OpenAttack: An Open-source Textual Adversarial Attack Toolkit. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 363–371.
[80] Hongming Zhang, Xinran Zhao, and Yangqiu Song. 2021. A Brief Survey and Comparative Study of Recent Development of Pronoun Coreference Resolution in English. In *Proceedings of the Fourth Workshop on CRAC*. ACL, Punta Cana, Dominican Republic, 1–11.
[81] Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019. Generating Fluent Adversarial Examples for Natural Languages. In *Proceedings of the 57th Conference of the ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 5564–5569.
[82] Wei Emma Zhang, Quan Z. Sheng, Ahoud Alhazmi, and Chenliang Li. 2020. Adversarial Attacks on Deep-Learning Models in Natural Language Processing: A Survey. *ACM Trans. Intell. Syst. Technol.* 11, 3, Article 24 (apr 2020), 41 pages.
[83] Zhi Quan Zhou and Liqun Sun. 2018. Metamorphic Testing for Machine Translations: MT4MT. In *Proceedings of the 25th Australasian Software Engineering Conference (ASWEC)*. IEEE Computer Society, Adelaide, SA, Australia, 96–100.
[84] Enwei Zhu and Jinpeng Li. 2022. Boundary Smoothing for Named Entity Recognition. In *Proceedings of ACL*. Association for Computational Linguistics, Dublin, Ireland, 7096–7108.
[85] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of ACL*. Association for Computational Linguistics, Sofia, Bulgaria, 434–443.

Received 2023-03-02; accepted 2023-07-27