S^2LCT: Specification- and Syntax-based Automated Testing Linguistic Capabilities of NLP Models

***Abstract*—Natural language processing has been widely used for developing a variety of text analytic applications with high reliability. For any NLP-based applications to be trustworthy, assessment through measuring the accuracy of the holdout datasets is needed. Recent research introduced methods that assess an NLP model on a set of linguistic capabilities. However, we observe that state-of-the-art methods rely on manually created test cases, which may be limited in terms of scalability and diversity. In this work, we introduce S^2LCT, an automated linguistic capability-based testing technique. Given a set of linguistic capabilities that assess an NLP model, S^2LCT first searches for suitable seed inputs from existing datasets, and then generates a sufficient number of new test inputs by expanding the seed inputs based on their context-free grammar (CFG). S^2LCT guarantees the correctness of the generated test input labels and that these test inputs belong to the correct linguistic capabilities. Evaluation results show that S^2LCT can generate test cases that are 100% more diverse than state-of-the-art while ensuring the correctness of labels and linguistic capabilities of the test inputs. We also show that S^2LCT facilitates the identification of critical failures and its origins in the NLP models.**

I. Introduction

Natural language processing (NLP) applications are growing rapidly. As a result, trustworthiness in how the quality of NLP applications is assessed becomes critical for its practical use in the real world. Traditionally, the quality of NLP models is assessed using aggregated metrics. In particular, accuracy (i.e., the fraction of outputs that the model correctly predicts) is the most widely used metric for assessing the quality of classification models: higher accuracy suggests better quality of a model. However, all NLP models have their strengths and weaknesses; using a single, aggregated metric (i.e., accuracy) makes it difficult for the users to assess the capabilities of NLP models. Such a metric fails to validate how well a model supports linguistic capabilities [1, 2] (e.g., when the model always fails on certain type of inputs).

To address this limitation, methodologies to assess an NLP model on a set of linguistic capabilities have been introduced [3, 4]. A linguistic capability explains a functionality of the input and output behavior for the NLP model under test. Typically, it describes certain type of inputs and outputs observed in real world for the target NLP task. Testing the linguistic capabilities allows the model developers to better understand the capabilities and potential issues of the NLP models. For example, CHECKLIST [3], a behavioral testing framework for evaluating NLP models, defines a linguistic capability called "Negated neutral should still be neutral" to measure how accurately a sentiment analysis model understands that the negated neutral input has neutral sentiment [3]. It requires the sentiment analysis model to output neutral sentiment on the negated neutral inputs. Such evaluation methodology avoids the overestimation of the model performance as it measures the model performance on each functionality. In the end, testing through linguistic capabilities provides not only the overall model performance, but also the malfunction facets of the model.

However, existing linguistic capability-based testing approaches rely on manually generated input templates, which need to be preset before generating tests. As a result, these templates are limited in their sentence structures. This restricts existing approaches' ability to comprehensively test linguistic capabilities.

To address this issue, we present S^2LCT, an automated linguistic capability-based testing framework for NLP models. The goal of S^2LCT is to generate a diverse linguistic capability-based test suite for a given model, both in terms of covering diverse linguistic structures and diverse model behaviors. To achieve the goal, S^2LCT needs to address two challenges. *(i) Capability-based Assessment.* Each test case should be automatically categorized into a linguistic capability; and *(ii) Automated Test Oracle.* The label of each test case should be automatically and accurately defined.

Capability-based Assessment. The suitability of test sentences for evaluating NLP models on a linguistic capability is determined by its relevancy to this linguistic capability. It is challenging to maintain relevancy between a test sentence and its linguistic capability when transforming and expanding the test sentence. This is because the linguistic capability is defined on a specific mixture of syntax and semantics of the sentence. Due to the inherent ambiguity of natural language sentences, there exists no automatic way to check the consistency of each sentence with the semantics specified in the corresponding linguistic capability. To address this difficulty, S^2LCT defines specifications (search rules and transformation templates) of linguistic capabilities, and analyzes the parse tree of the seed sentence to identify possible expansion.

Automated Test Oracle. For NLP model testing, test oracle is usually determined by understanding the semantics of texts and requires domain knowledge of different NLP tasks. Thus, the current testing practice requires manual efforts to create the test oracles of the holdout data. The manual work is costly in terms of time consumption. Therefore, it necessitates automated test oracle generation for improving the testing process of NLP models. However, automatically generating the test oracles remains one of the main challenges in NLP software testing [5]. A few metamorphic testing approaches have been proposed

in image recognition domain, but they require understanding the characteristics of metamorphic relations between inputs and outputs, which require domain expertise and non-trivial manual efforts [6]. In addition, it is even more challenging to design metamorphic relations in textual data because the semantics of nature language sentences can be greatly changed even by a slight perturbation to the sentences.

In this work, as a first step, we consider sentiment analysis as the NLP task for the models under test. S^{2}LCT obtains the appropriate test oracle by implementing domain-specific knowledge on the word sentiment dataset and word suggestion model for validating the generated sentence and its oracle.

We demonstrate the generality of S^{2}LCT and its utility as a NLP model evaluation tool by evaluating three well-known sentiment analysis models: BERT-base [7], RoBERTa-base [8] and DistilBERT-base [9].

We made the following contributions in this work:

- We design and implement an automated testing tool, S^{2}LCT, and compared it with CHECKLIST. We find that the test cases generated by S^{2}LCT achieve 100% higher coverage than CHECKLIST when used for testing sentiment analysis models.
- We perform a manual study to measure the correctness of the sentiment labels and their linguistic capabilities produced by S^{2}LCT. We find that S^{2}LCT generates test cases that consistently label their sentiment correctly as human.
- We analyze the root causes of misclassification in the sentiment analysis models, guided by S^{2}LCT testing result. We find that seeds and expanded test cases produced by S^{2}LCT are rather useful in helping developers understand bugs in the model.

II. Background

In this section, we provide a brief background of CHECKLIST's [3] approach of test case generation via an example. CHECKLIST introduces task-dependent linguistic capabilities for monitoring model performance on each linguistic capability. It assumes that the linguistic phenomena can be represented in the behavior of the model under test, because each linguistic capability specifies the desired behavior between model inputs and outputs. For each linguistic capability, CHECKLIST manually creates test case templates and generates sentences by filling in the values of each placeholder.

We show an example of CHECKLIST templates in Figure 1. These templates are used for evaluating the sentiment analysis model on the linguistic capability of "Sentiment change over time, present should prevail". For the templates defined at lines 10 to 16, they have placeholders such as it, air_noun, pos_adj. Values for the placeholders are defined at lines 1 to 9. For each template, CHECKLIST fills in all the combinations of the values of placeholders to generate sentences under this linguistic capability. For example, sentences such as "The flight is good" and "That airline was happy" are generated using the template at line 10.

```
1  air_noun = [
2      'flight', 'seat', 'pilot', 'staff', ...
3  ]
4  pos_adj = [
5      'good', 'great', 'excellent', 'amazing', ...
6  ]
7  neg_adj = [
8      'awful', 'bad', 'horrible', 'weird', ...
9  ]
10 t = editor.template('{it} {air_noun} {be} {pos_adj}.',
11                     it=['The', 'This', 'That'], be=['is', 'was'],
                           labels=2, save=True)
12 t += editor.template('{it} {be} {a:pos_adj} {air_noun}.',
13                     it=['It', 'This', 'That'], be=['is', 'was'],
                           labels=2, save=True)
14 t += ...
15 t += editor.template('{it} {be} {a:neg_adj} {air_noun}.',
16                     it=['It', 'This', 'That'], be=['is', 'was'],
                           labels=0, save=True)
```

Fig. 1: Example of CHECKLIST templates on the linguistic capability of "Sentiment change over time, present should prevail".

Despite the simple approach used for test case generation, CHECKLIST has limitations. First, it relies on manual work for defining template structure and its placeholder values, making the test case generation a costly process. Moreover, such manual work generates a limited number of templates, produces similar test cases, and induces bias in the test cases. These limitations motivated the design of our approach.

III. Specification- and Syntax-based Linguistic Capability Testing

We design and implement a new NLP model testing method, *Specification- and Syntax-based Linguistic Capability Testing (S^{2}LCT)*, which automatically generates test cases with oracles to assess the quality of NLP models.

Figure 2 depicts an overview of S^{2}LCT, which consists of two phases. The *specification-based seed generation* phase performs rule-based searches from a real-world dataset and template-based transformation to obtain the initial seed sentences. The search rules (e.g., search for neutral sentences that do not include any positive or negative words) and transformation templates (e.g., negating a sentence) are defined in the *linguistic capability specifications* (Table I). Using the specification will result in seed sentences that are most likely to conform to a specific linguistic capability and is labelled correctly.

The *syntax-based sentence expansion* phase expands the seed sentences with additional syntactic elements (i.e., words) to cover many more real-world syntactic structures. It first performs a syntax analysis to identify the part-of-speech (PoS) tags that can be inserted to each seed, by comparing the PoS parse trees between the seed sentence and many other sentences from a large reference dataset. Each identified tag is inserted into the seed as a *mask*. It then uses an NLP recommendation model (i.e., BERT [7]) to suggest possible words. Finally, the resulting sentence is validated to be consistent with the

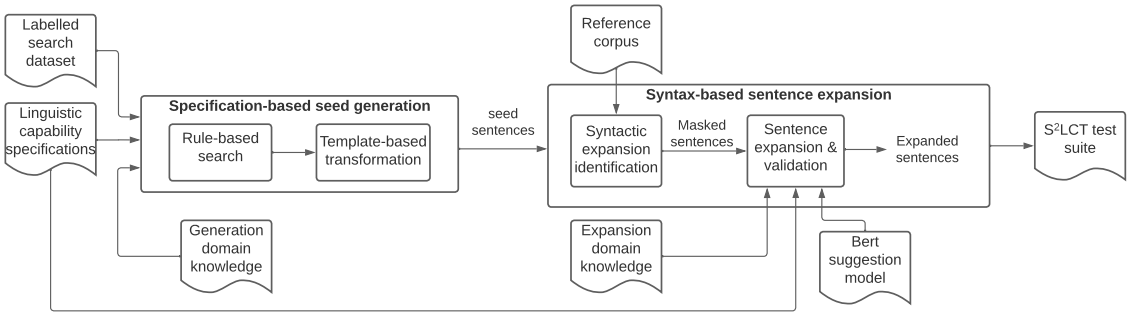

Fig. 2: Overview of S^{2}LCT.

specification which additionally defines the rules for expansion (e.g., the expanded word should be neutral).

We now describe each phase of S^{2}LCT in detail.

A. Specification-based Seed Generation

The seed generation phase of S^{2}LCT starts by searching sentences in a real-world dataset, which match the rules defined in the linguistic capability specification, and then transforming the matched sentences using templates to generate seed sentences that conform to individual linguistic capabilities. The reasons for this design choice are twofold. First, while it is generally infeasible to judge which linguistic capability any sentence falls into and which label it should have, there exist simple rules and templates that allow for classifying the resulting sentences into individual linguistic capabilities and with the correct labels, with high confidence. This enables us to test each linguistic capability individually. Second, searching from a real-world dataset ensures that the sentences used as test cases for testing linguistic capabilities are realistic and diverse. The diverse test cases are more likely to achieve a high coverage of the target model's functionality in each linguistic capability, thus detecting more errors.

Specifically, S^{2}LCT first searches and selects sentences applicable to the linguistic capability in a given real-world dataset with search rules. In case the search rules only fulfill part of the linguistic capability specifications (i.e., LC3-LC10 in Table I), we then transform the selected sentences into seed sentences using the heuristic templates.

Table I shows the search rules and the transformation templates of all 10 linguistic capabilities we implemented in S^{2}LCT. The first column shows the linguistic capability type and its description, and the second column shows the search rule and transformation template used in each linguistic capability. For LC1 and LC2, the NLP models are evaluated in the scope of short sentences with selective sentiment words. It does not require any transformation because the search rules alone are sufficient to find sentences that conform to the linguistic capabilities. On the other hand, search rules of LC3 to LC10 are not enough to match their linguistic capability specification; thus, S^{2}LCT additionally uses templates to transform the searched sentences to match the corresponding linguistic capability. For example, in LC3's transformation template, the searched sentences are appended as part of a sentence to generate a seed. In LC4's transformation template, the searched demonstrative sentences are negated.

B. Syntax-based Sentence Expansion

The simple search rules and transformation templates used to generate the seed sentences may limit the syntactic structures these seeds may cover. To address this limitation, the syntax-based sentence expansion phase extends the seed sentences to cover syntactic structures commonly used in real-life sentences. Our idea is to differentiate the parse trees between the seed sentences and the reference sentences from a large real-world dataset. The extra PoS tags in the reference parse trees are identified as potential syntactic elements for expansion and inserted into the seed sentences as masks. We then use masked language model to suggest the fill-ins. If the resulting sentences still conform to the linguistic capability specification, they are added to S^{2}LCT's test suite.

1) Syntax Expansion Identification: Algorithm 1 shows how masks are identified for each seed sentence. It takes the parse trees of the seeds, generated by the Berkeley Neural Parser [10, 11], and a reference context-free grammar (CFG) from the Penn Treebank corpus dataset [12] as inputs. This reference CFG was learned from a large dataset [13] that is representative of the distribution of real-world language usage. The algorithm identifies the discrepancy between the seed syntax and the reference grammar to decide how a seed can be expanded.

For each production in each seed's parse tree (lines 3 and 4), we extract its non-terminal at the left-hand-side (line 5), s_lhs, and the grammar symbols at the right-hand-side (line 6), s_rhs. In line 7, the algorithm iterates through all productions in the reference context-free grammar and matches these that have the same non-terminal at the left-hand-side as s_lhs. The right-hand-side of each matched production is called r_rhs. If s_rhs consists of a subset of the grammar symbols in r_rhs (line 8), the additional symbols in the r_rhs are inserted as masks in the parse tree of the seed sentence, in their respective positions in the expanded production. The left to right traversal of the leaves of an expanded parse tree forms a masked sentence. Due to a potential large number of masked sentences created by this algorithm, we randomly select k masked sentences for the next sentence expansion and validation phase (line 14).

TABLE I: Search rules and transformation templates for linguistic capabilities.

Linguistic capability	**Search rule and transformation template**
LC1: Short sentences with neutral adjectives and nouns	**Search** seed={length: $<$ 10; include: neutral adjs & neutral nouns; exlcude: pos adjs & neg adjs & pos nouns & neg nouns; label: neutral} **Transform** N/A
LC2: Short sentences with sentiment-laden adjectives	**Search** seed={length: $<$ 10; include: pos adjs; exlcude: neg adjs & neg verbs & neg nouns; label: pos} | {length: $<$ 10; include: neg adjs;exclude: pos adjs & pos verbs & pos nouns & neg verbs & neg nouns; label: neg} **Transform** N/A
LC3: Sentiment change over time, present should prevail	**Search** pos_sent={label: pos, length: $<$ 20}, neg_sent={label: neg, length: $<$ 20} **Transform** seed={[‘Previously, I used to like it saying that’,‘Last time, I agreed with saying that’,‘I liked it much as to say that’]+[pos_sent | neg_sent]+[‘but’, ‘although’, ‘on the other hand’]+[‘now I don’t like it.’, ‘now I hate it.’]} | {[‘I used to disagree with saying that’,‘Last time, I didn’t like it saying that’,‘I hated it much as to say that’]+[neg_sent, pos_sent]+[‘but’, ‘although’, ‘on the other hand’]+[‘now I like it.’]}
LC4: Negated negative should be positive or neutral	**Search** demonstrative_sent={start: [This, That, These, Those] + [is, are]; label: neg} **Transform** seed=negation of demonstrative_sent ([‘is’] $\rightarrow$ [‘is not’, ‘isn’t’], [‘are’] $\rightarrow$ [‘are not’, ‘aren’t’])
LC5: Negated neutral should still be neutral	**Search** demonstrative_sent={start: [This, That, These, Those] + [is, are]; label: neutral} **Transform** negation of demonstrative_sent
LC6: Negation of negative at the end, should be positive or neutral	**Search** neg_sent={label: neg} **Transform** seed={[‘I agreed that’, ‘I thought that’]+[neg_sent]+[‘but it wasn’t’, ‘but I didn’t’]}
LC7: Negated positive with neutral content in the middle	**Search** pos_sent={length: $<$ 20; label: pos}, neutral_sent={length: $<$ 20; label: neutral} **Transform** seed={[‘I wouldn’t say,’, ‘I do not think,’, ‘I don’t agree with,’]+[neutral_sent]+[‘,’]+[pos_sent]}
LC8: Author sentiment is more important than of others	**Search** pos_sent={label:pos}, neg_sent={label: neg} **Transform** seed={[temp1]+[pos_sent]+[temp2]+[neg_sent]} | {[temp1]+[neg_sent]+[temp2]+[pos_sent]} where temp1={[‘Some people think that’, ‘Many people agree with that’, ‘They think that’, ‘You agree with that’], temp2=[‘but I think that’]}
LC9: Parsing sentiment in (question, yes) form	**Search** pos_sent={label: pos}, neg_sent={label: neg} **Transform** seed={[‘Do I think that’, ‘Do I agree that’]+[pos_sent | neg_sent]+[‘? yes’]}
LC10: Parsing sentiment in (question, no) form	**Search** pos_sent={label:pos}, neg_sent={label:neg} **Transform** seed={[‘Do I think that’, ‘Do I agree that’]+[pos_sent neg_sent]+[‘? no’]}

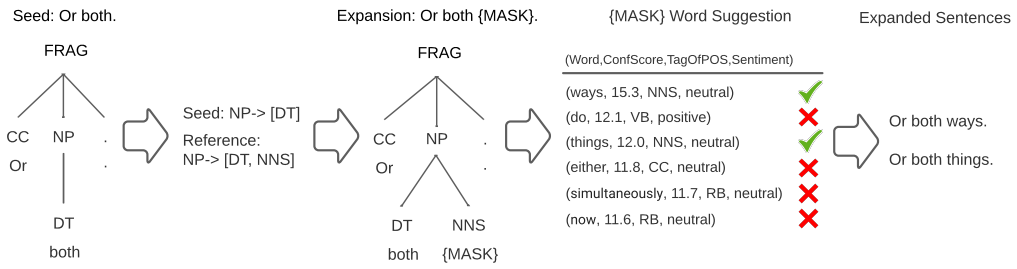

Fig. 3: Example of sentence expansion.

Running example.: Figure 3 shows an example using Algorithm 1 to generate a masked sentence. The sentence “Or both.” is a seed of the linguistic capability of “Short sentences with neutral adjectives and nouns”. The tree on the left shows the parse tree of this seed; it consists of two productions: “$FRAG \rightarrow [CC, NP, .]$” and “$NP \rightarrow [DT]$”. When matching the left-hand-side non-terminal of the second production (i.e., “NP”) in the reference CFG, we found that it includes a production “$NP \rightarrow [DT, NNS]$” which has an additional symbol “NNS” on the right-hand-side. The algorithm thus expands the parse tree with this symbol, shown in the second tree. The masked sentence “Or both {MASK}.” is the result of the left-to-right traversal of this expanded parse tree.

2) Sentence Expansion and Validation: In this phase, the words to fill in the masks in the masked sentences are suggested by the BERT model [7]. BERT is a transformer-based natural language model that is pre-trained on two tasks: masked token prediction and next sentence prediction. BERT model is capable of suggesting words for the masked token according to its surrounding context in a sentence. For each masked token, multiple words may be suggested, ranked by their confidence scores. Because BERT model is not aware of the linguistic capability specification and the grammar symbol in the expanded parse tree, an expanded sentence using the suggested words may no longer satisfy the linguistic capability specification. Therefore, we perform validation on the suggested

Algorithm 1 Syntax expansion identification algorithm.

Input: Parse trees of seed sentences S, reference context-free grammar R
Output: Set of masked sentences M
for each part tree s from S **do**
  for each production s_prod from s **do**
    $s\_lhs = s\_prod.lhs$
    $s\_rhs = s\_prod.rhs$
    for each r_rhs from $R[s\_lhs]$ **do**
      if $s\_rhs \subset r\_rhs$ **then**
        $M = M \cup insertMask(\text{r\_rhs-s\_rhs}, s)$
      end if
    end for
  end for
end for

return $random(M, k)$

words and only accept them if the following three criteria are met.

First, the PoS tag of the suggested word must match the PoS tag of the expanded symbol in the parse tree. For the example in Figure 3, the masked symbol is a "NNS" (i.e., plural noun); thus, the suggested word must also be a "NNS". We use SpaCy [14], a free open-source library for natural language processing, to extract the PoS tag for each suggested word. Second, the sentiment of the expanded sentence must be the same as the seed sentence. To ensure this, we require the suggested words be neutral. Third, we additionally verify that the expanded sentences satisfy the same search rules for the seed sentences in LC1 and LC2. This criteria cannot be applied to other linguistic capabilities because they have additional transformation templates.

Running example.: The third step in Figure 3 shows the words suggested by BERT. For this masked sentence, BERT suggested six words. Each word is associated with the confidence score provided by BERT, the PoS tag, and the sentiment. Among the six words, only "ways" and "things" are validated by S^{2}LCT because they have the Pos tag "NNS" and are neutral. In addition, both sentences still meet the search rule of the associated linguistic capability of "Short sentences with neutral adjectives and nouns". In the end, the syntax-based sentence expansion results in two sentences, "Or both ways." and "Or both things.", from the seed "Or both.".

IV. Experimental Setup

In this section, we present the setup of our experiments to evaluate the effectiveness of S^{2}LCT. We answer the following research questions (RQs):

RQ1 : **Test Case Diversity.** Can S^{2}LCT generate more diverse test cases than CHECKLIST?

RQ2 : **Test Oracle.** Can S^{2}LCT generate test sentences with correct sentiment labels?

RQ3 : **Capability Testing.** Are S^{2}LCT generated test sentences correctly categorized into a linguistic capability?

A. Experimental Subjects

NLP Models & Dataset: We evaluate our approach on three learning-based sentiment analysis models released from the Hugging Face centralized model hub:[1] `BERT-base` (textattack/bert-base-uncased-SST-2), `RoBERTa-base` (textattack/bert-base-uncased-SST-2), and `DistilBERT-base` (distilbert-base-uncased). These models were pre-trained on English language using a masked language modeling (MLM) objective, and were fine-tuned on the sentiment analysis task. In this experiment, we used the SST-2 [15] dataset for searching the seeds in S^{2}LCT. SST-2 is a corpus of movie review, consisting of 11,855 sentences with sentiment scores. We split the scores into ranges [0, 0.4], (0.4, 0.6] and (0.6, 1.0] to assign them negative, neutral and positive labels, respectively. In addition, SentiWordNet [16] is a publicly available dataset for English sentiment lexicons. It provides lexical sentiment scores and the sentiment word labels are categorized by implementing the rules. We used SentiWordNet as both the generation and the expansion domain knowledge as shown in Figure 2.

Comparison Baseline: We compared S^{2}LCT with CHECKLIST,[2] a manual template and linguistic capability based approach to generate test cases. In this evalaution, we used CHECKLIST's sentiment analysis test cases that are generated from its publicly available Jupyter Notebook implementation.

B. Experimental Process

RQ1: Recall that CHECKLIST relies on significant manual efforts and may not generate comprehensive test cases in a linguistic capability. S^{2}LCT, instead, automatically generates test cases based on a search dataset and the syntax in a large reference corpus. We expect S^{2}LCT can generate a more diverse test suite than CHECKLIST.

We first evaluate the three sentiment analysis models by testing them on the S^{2}LCT test cases, reporting number of test cases and each model's failure rate in each linguistic capability. Specifically, for each linguistic capability, we randomly choose 50, 100 and 200 seeds , and expand test cases from these seeds. We selected subsets of seeds to be used for expansion because the syntax-based expansion phase may be expensive to run all the generated seeds. We tested the three models using both seed and expanded test cases. To account for the randomness in random seed selection, we repeated these experiments 3 times and report median number of test cases and each model's failure rate.

We use two metrics to compare the diversity between the S^{2}LCT generated seeds and the CHECKLIST test cases.

Self-BLEU: We reuse the input diversity metric, called Self-BLEU, that Zhu et al. introduced [17]. BLEU evaluates the token-level similarity. The Self-BLEU is defined as the average BLEU scores over all reference sentences. A higher Self-BLEU score indicates less diversity in the test suite. In the

[1]https://huggingface.co/models
[2]https://github.com/marcotcr/checklist

experiment, we collected 50, 100 and 200 randomly selected S^{2}LCT seeds and reported the median Self-BLEU score over three trials for each group of seeds.

Production rule coverage.: We propose a new metric to evaluate the syntactic diversity of the generated test suite. It is defined as the number of production rules used in a set of test sentences. In our experiments, we used the Berkeley Neural Parser [10, 11] to parse and collect all the production covered in a set of test sentences. We compared the production rule coverage between 50, 100 and 200 randomly selected S^{2}LCT seeds and the CHECKLIST test cases.

In addition, we follow the approach presented by Ma et al. [18], where the authors measure the coverage of NLP model intermediate states as corner-case neurons. Because the matrix computation of intermediate states impacts NLP model decision-making, a test suite that covers a greater number of intermediate states can represent more NLP model decision-making, making it more diverse. Specifically, we used two coverage metrics by Ma et al. [18], *boundary coverage* (BoundCov) and *strong activation coverage* (SActCov), to evaluate the test suite diversity. It is worth noting that a test sample with a statistical distribution similar to the training data is rarely found in the corner case region. Thus, covering a larger corner case region indicates that the test suite is more likely to be buggy.

$$\begin{aligned}\text{UpperCorner}(\mathcal{X}) &= \{n \in N | \exists x \in \mathcal{X} : f_n(x) \in (high_n, +\infty)\};\\ \text{LowerCorner}(\mathcal{X}) &= \{n \in N | \exists x \in \mathcal{X} : f_n(x) \in (-\infty, low_n)\};\end{aligned} \tag{1}$$

Equation 1 defines the corner-case neuron of the NLP model $f(\cdot)$, where $\mathcal{X}$ is the given test suite, N is the number of neurons in model $f(\cdot)$, $f_n(\cdot)$ is the n^{th} neuron's output, and $high_n$ and low_n are the n^{th} neurons' output bounds on the model training dataset. Equation 1 can be interpreted as the collection of neurons that emit outputs beyond the model's numerical boundary.

$$\begin{aligned}BoundCov(\mathcal{X}) &= \frac{|UpperCorner(\mathcal{X})| + |LowerCorner(\mathcal{X})|}{2 \times |N|}\\ SActCov(\mathcal{X}) &= \frac{|UpperCorner(\mathcal{X})|}{|N|}\end{aligned} \tag{2}$$

The definition of our coverage metrics is shown in Equation 2, where BoundCov measures the coverage of neurons that produces outputs exceeding the upper or lower bounds, and SActCov measures the coverage of neurons that creates outputs exceeding the lower bound. Higher coverage indicates the test suite is better for triggering the corner-case neurons, thus better test suite diversity.

To answer RQ1, for each NLP model under test, we first feed its training dataset to compute each neuron's lower and upper bounds. After that, we select the same number of test cases from S^{2}LCT and CHECKLIST as the test suite and compute the corresponding coverage metrics.

RQ2 and RQ3: As described in Section III, S^{2}LCT generates test cases in two steps: specification-based seed generation and syntax-based sentence expansion. These automated steps may generate seed/expanded sentences marked with incorrect sentiment labels or categorized into wrong linguistic capabilities. To answer RQ2 and RQ3, we performed a manual study to measure the correctness of the sentiment labels and linguistic capabilities associated with the seed/expanded sentences, produced by S^{2}LCT.

In the manual study, we randomly sample 100 S^{2}LCT seed sentences, each of which has at least one expanded sentence, and divide these seeds to two sets (i.e., 50 in each set). For each sampled seed sentence, we randomly obtain one of its expanded sentences. This forms the two sets of sentences (200 sentences in total) we use for this study, each with 50 seeds and 50 corresponding expanded sentences. We recruited three participants for this study; all of them are graduate students with no knowledge about this work. 2 of them were assigned one set of sentences and the third was assigned the other set. Each participant was asked to provide two scores for each sentence. *(1) Relevancy score between sentence and its associated linguistic capability*: this score measures the correctness of S^{2}LCT linguistic capability categorization. The scores are discrete, ranging from 1 ("strongly not relevant") to 5 ("strongly relevant"). *(2) sentiment score of the sentence*: this score measures the sentiment level of the sentence . It is also discrete, ranging from 1 ("strongly negative") to 5 ("strongly positive"). We measure the following:

$$Label\_consistency = \frac{1}{\#Sample} \cdot \sum_i \delta(label_{S^2LCT} = label_{human}) \tag{3}$$

$$LC\_relevancy_{AVG} = \frac{1}{\#Sample} \cdot \sum_i Norm(LC\_relevancy_i) \tag{4}$$

Equation 3 represents the percentage of the test cases that S^{2}LCT and the participants produce the same sentiment labels. High value of this metric indicates S^{2}LCT generates test cases with correct labels. Equation 4 represents the average of the normalized relevancy score between a sentence and its associated linguistic capability. The relevancy score is to evaluate the relevancy of the sentence to be used for testing model under test on the corresponding linguistic capability. Higher average score means indicate the linguistic capability categorization by S^{2}LCT is correct. We answer RQ2 and RQ3 using the metrics defined by Equation 3 and Equation 4, respectively.

V. Experimental Results

This section presents the experimental results and the answers to the RQs. More results are available at https://github.com/csresearcher27/s2lct_project.

TABLE II: Results of BERT-base, RoBERTa-base and DistilBERT-base sentiment analysis models on S^{2}LCT test cases using 50 seeds. BERT-base, RoBERTa-base and DistilBERT-base models are denoted as BERT, RoBERTa and dstBERT, respectively.

Linguistic capability	**S^{2}LCT #Seeds**	**S^{2}LCT #Exps**	**S^{2}LCT #Fail**	**S^{2}LCT Fail rate[%]**	**S^{2}LCT #PassTo-Fail**
LC1: Short sentences with neutral adjectives and nouns	19	224	BERT: 213 RoBERTa: 192 dstBERT: 233	BERT: 87.36 RoBERTa: 79.01 dstBERT: 96.28	BERT: 17 RoBERTa: 9 dstBERT: 0
LC2: Short sentences with sentiment-laden adjectives	50	467	BERT: 6 RoBERTa: 6 dstBERT: 5	BERT: 1.16 RoBERTa: 1.21 dstBERT: 0.86	BERT: 6 RoBERTa: 3 dstBERT: 5
LC3: Sentiment change over time, present should prevail	50	1523	BERT: 226 RoBERTa: 723 dstBERT: 953	BERT: 13.52 RoBERTa: 43.27 dstBERT: 60.58	BERT: 52 RoBERTa: 31 dstBERT: 60
LC4: Negated negative should be positive or neutral	50	2073	BERT: 1907 RoBERTa: 1828 dstBERT: 1827	BERT: 89.83 RoBERTa: 87.08 dstBERT: 86.06	BERT: 58 RoBERTa: 51 dstBERT: 34
LC5: Negated neutral should still be neutral	26	1090	BERT: 1039 RoBERTa: 1038 dstBERT: 1073	BERT: 93.01 RoBERTa: 93.31 dstBERT: 95.33	BERT: 61 RoBERTa: 41 dstBERT: 83
LC6: Negation of negative at the end, should be positive or neutral	50	1345	BERT: 1384 RoBERTa: 1384 dstBERT: 1382	BERT: 99.95 RoBERTa: 99.21 dstBERT: 99.07	BERT: 0 RoBERTa: 0 dstBERT: 0
LC7: Negated positive with neutral content in the middle	50	1137	BERT: 934 RoBERTa: 831 dstBERT: 972	BERT: 89.86 RoBERTa: 70.01 dstBERT: 92.95	BERT: 6 RoBERTa: 26 dstBERT: 4
LC8: Author sentiment is more important than of others	50	1659	BERT: 331 RoBERTa: 293 dstBERT: 420	BERT: 21.92 RoBERTa: 15.92 dstBERT: 33.68	BERT: 51 RoBERTa: 47 dstBERT: 49
LC9: Parsing sentiment in (question, yes) form	50	1319	BERT: 91 RoBERTa: 81 dstBERT: 82	BERT: 6.65 RoBERTa: 6.14 dstBERT: 5.99	BERT: 20 RoBERTa: 27 dstBERT: 13
LC10: Parsing sentiment in (question, no) form	50	1417	BERT: 1181 RoBERTa: 1117 dstBERT: 1112	BERT: 79.46 RoBERTa: 82.74 dstBERT: 82.37	BERT: 27 RoBERTa: 13 dstBERT: 4

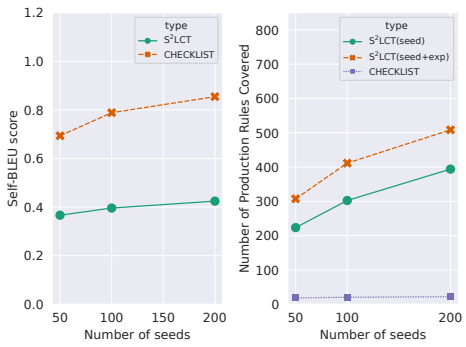

Fig. 4: Results of Self-BLEU (left) and Production Rule Coverage (right) of S^{2}LCT and CHECKLIST test cases.

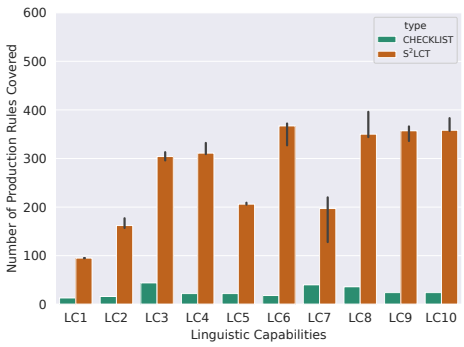

Fig. 5: Results of production rule coverage between S^{2}LCT with 50 seeds and CHECKLIST test cases.

A. RQ1: Test Case Diversity

Our results show that *S^{2}LCT generated many test cases that the sentiment analysis models failed to predict the correct labels, and it produced significantly more diverse test cases than CHECKLIST did.*

Self-BLEU: Left in Figure 4 compares the Self-BLEU scores between S^{2}LCT seed sentences and CHECKLIST test cases. The x-axis shows sizes of random samples of S^{2}LCT seeds and CHECKLIST test cases, and y-axis shows the Self-BLEU scores. Median of the Self-BLEU scores over all linguistic capabilities is shown in left in figure 4. We observe that Self-BLEU scores of CHECKLIST test cases is significantly higher than those of S^{2}LCT seeds, for all numbers of seeds. This result indicates CHECKLIST test cases are less diverse than S^{2}LCT seeds, which demonstrates the benefit of searching from a real-world search dataset over creating test cases from limited number of preset templates.

Production Rule Coverage: Right in figure 4 compares production rule coverage between the S^{2}LCT seed sentences, S^{2}LCT seed and expanded sentences, and the CHECKLIST test cases. The x-axis shows the sizes of random samples of S^{2}LCT seed sentences, S^{2}LCT seed and expanded sentences, and CHECKLIST test cases, and y-axis shows the production rule coverage scores. Median of the production rule coverage

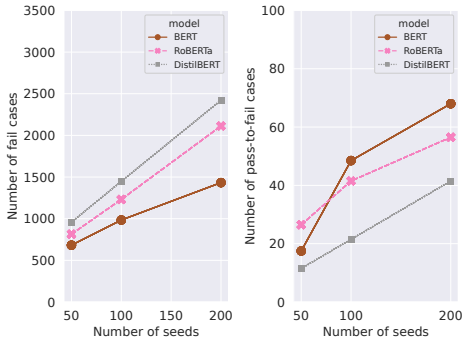

Fig. 6: Number of failure (left) and Pass-to-Fail (right) results of NLP models on S^{2}LCT test cases over numbers of seeds.

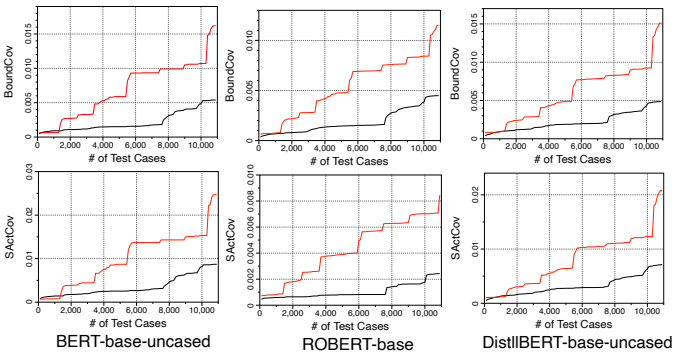

Fig. 7: Coverage results of S^{2}LCT and CHECKLIST test cases.

scores over all linguistic capabilities is shown in the right in the figure 4. The figure 4 shows that S^{2}LCT seed and/or expanded sentences produce significantly higher production rule coverage scores than CHECKLIST test cases did. Also, more S^{2}LCT seeds cover more production rules.

Figure 5 compares the production rule coverage between 50 S^{2}LCT seeds and all CHECKLIST test cases for each linguistic capability. The x-axis shows each one linguistic capability, and the y-axis is the production rule coverage for these test cases. We observed that even with only 50 seeds in each linguistic capability, S^{2}LCT seeds cover significantly higher number of production rules than all CHECKLIST test cases (ranging from 95 for LC1 to 367 for LC6 for S^{2}LCT seeds and from 13 for LC1 to 44 for LC3 for CHECKLIST seeds) in each linguistic capability.

Overall, the above results show that S^{2}LCT test cases are significantly more diverse in terms of syntactic structure than CHECKLIST.

Model Test Results: Table II shows the testing results of the three NLP models on S^{2}LCT test cases using 50 seeds. First column lists linguistic capabilities for the sentiment analysis task, and Column 2 shows the numbers of seed test cases, and Column 3 shows the median numbers of expanded test cases over 3 trials. Columns 4 and 5 show the median numbers of failed test cases and the failure rates (i.e., percentage of test cases that a model predicts incorrect labels) of each NLP models on the seed and expanded test cases over the 3 trials, respectively. Column 6 shows the median number of expanded test cases that failed, but their corresponding seed test cases passed over the 3 trials(Pass-to-Fail). We observe that in all linguistic capabilities, S^{2}LCT produces hundreds of test cases, expanding at least an order of magnitude more test cases than the seeds. LC1 and LC5 have 19 and 26 seeds, respectively. This is because S^{2}LCT's search rules and transformation templates of these two linguistic capabilities produced few seeds. Nevertheless, the syntax-base sentence expansion phase generated of 224 and 1090 test cases, respectively. In terms of model performance, all three models achieve low failure rates in LC2 and LC9 while the failure rates are high in all other linguistic capabilities (13.52%-99.95%). We also observe that there are many expanded test cases that failed, but their corresponding seeds did not (last column). This shows that the syntax-based sentence expansion phase indeed introduces more diverse sentence structures in the test cases that cause the models to fail.

In addition, Figure 6 shows the numbers of fail and pass-to-fail cases with different number of seeds used. We observed that more S^{2}LCT seeds introduce more failed cases and more Pass-to-Fail cases, demonstrating the benefit of using more seeds when resource permits.

Model Coverage: Figure 7 shows the coverage results of S^{2}LCT and CHECKLIST test cases. The red line represents S^{2}LCT coverage and the black line represents CHECKLIST coverage. Each column in Figure 7 represents the results for one NLP model. The first row is the *BoundCov* results and the second row is the *SActCov* results.

We made three observations. First, for *all* experimental settings (i.e., NLP model and coverage metric), S^{2}LCT achieves higher coverage than CHECKLIST. Recall that a higher coverage implies the test cases are more diverse and do not have a similar statistical distribution to the model training data. As a result, a test suite with greater coverage complements the model training data distribution (i.e. holdout testing data) better. For example, for the first NLP model under test, S^{2}LCT can achieve a higher coverage than CHECKLIST with only half the number of test cases. This result confirms that S^{2}LCT can generate more diverse test cases to complement the holdout dataset for testing NLP models. Second, as the number of test cases increases, the test suite can achieve better coverage. Such observation is intuitive. However, generating a more extensive test suite is not easy, particularly for CHECKLIST, which is a manually template-based approach. Third, for each NLP model, there is no fixed relationship between *BoundCov* and *SActCov*. In other words, while a test suite may produce higher *BoundCov* for some models, the same test suite may get higher *SActCov* for other NLP models. Recall that *BoundCov* measures both the upper and lower corner neurons and *SActCov* measures only the upper corner neurons. Such observation implies that the upper and lower corner neurons are distributed unevenly, and measuring only one of them is not enough.

TABLE III: Manual study results of label consistency and linguistic capability relevancy of S^{2}LCT generated test cases compared to human annotations.

Type	#Test_Cases	Label_Consistency	LC_Relevancy
Seed	100	0.83	0.90
Expanded	100	0.84	0.90

B. RQ2: Correctness of Sentiment Labels

Table III shows results of our manual study. The first column distinguishes the seed and expanded sentences. The number of test cases used for the study is represented in the second column. The label consistency score defined in Equation 3 is shown in column 3.

We observe that *S^{2}LCT generates test cases that consistently label their sentiment correctly.* Column 3 shows that the label consistency scores are 0.83 and 0.84 for the seed and expanded sentences, respectively. This means that S^{2}LCT generates test oracles consistent with human understanding most of the time. Also, there is little difference of the scores between the seed and expanded sentences. This implies that the syntax-based sentence expansion in S^{2}LCT preserves the sentiment as its seed.

C. RQ3: Correctness of Linguistic Capability Categorization

The linguistic capability relevancy score defined in Equation 4 is shown in Column 4 of Table III. The result shows that *S^{2}LCT generates test cases that are correctly categorized to the corresponding linguistic capabilities most of the time.* The linguistic capability relevancy scores for the seed and expanded sentences are both 0.9, achieving high order of agreement with human assessment. The fact that the expanded sentences generated by S^{2}LCT also have the same level of linguistic capability relevancy as the seed sentences shows that the syntax-based sentence expansion retains the linguistic capabilities.

VI. Application of S^{2}LCT

In this section, we use one case study to show S^{2}LCT can be useful to help developers to find root causes of bugs in the sentiment analysis models.

Experimental Process. we conduct experiments to demonstrate that S^{2}LCT can help developers understand the bugs in the NLP models. Recall that S^{2}LCT generates test cases by mutating seed sentences (e.g. by expanding one token in the seed input). Still, it is unclear why mutating one token will cause the model to produce misclassified results. We seek to help developers understand why such mutation will result in the misclassification. Existing work [19, 20, 21] has demonstrated that the ML model prediction is dominated by a minimal set of input features (i.e. tokens in input sentences). Motivated by such intuition, we identify a minimal set of input tokens that dominate the model prediction.

Formally, given a input sentence $x = [tk_1, tk_2, \cdots, tk_n]$, and the NLP model under test $f(\cdot)$, our goal is to find a masking template $T = [t_1, t_2, \cdots, t_n]$, where t_i is 0 or 1, representing masking the i^{th} token (i.e. tk_i) in x or not. The template T can mask some tokens in x with attribute tokens, and the masked input has a high probability of retaining the original prediction x, denoted as

$$P(f(T(x)) = f(x)) \geq P_{thresh} \tag{5}$$

To create such a template T, we first compute the contribution score of each input token using an existing explainable ML technique [20]. We then begin with the full mask template (i.e., all tokens are masked); such full mask template definitely does not satisfy Equation 5. We then iteratively shift one position from mask to non-mask based on the order of each token's contribution score, until the template T satisfies Equation 5. Because we iterate the size of the mask, the generated template T will keep the minimum number of tokens in x. Moreover, since the input x is an incorrect prediction, the generated template T is likely to produce misclassification (i.e., the probability to be misclassified is larger than P_{thresh}).

Results. We generate a template that dominates the NLP model prediction to assist developers in understanding the false predictions. Figure 8 shows the generated templates for two randomly selected seeds and their corresponding expanded test inputs. The first example tests "sentiments in (question, yes) form" (LC9), and the second example tests "negative positive with neutral content in the middle" (LC7). The first column shows the seed sentence, the second shows the expanded sentence, and the third shows each token's contribution score. The blue bar indicates the score for seed inputs, whereas the orange bar reflects the score for the expanded sentences. We highlight the mutated token with yellow background and generated templates with red text.

The results show that after mutating the seed sentence with one token, the token set that dominates the NLP model prediction has changed. We can trace the root causes to the bias of the training data on the linguistic capability under test; as a result, for the linguistic capability under test, the model has a bias towards positive/negative for certain token sequence patterns. For example, LC9 has a bias toward the token sequence pattern that includes "maybe .... but ... even... yes". Thus, adding the token "even" to the seed sentence will match one of those biased sequence patterns. Sentences with such pattern in the training dataset are dominantly positive; thus, the models make the wrong decision on the sentence with "even" as positive. The visualization of each token's contribution score in the third column confirms our observation. Once "even" is added, scores of other tokens such as "but" and "yes" all change from negative to positive. To fix the issue for LC9, we need to add more negative training samples with the format of "maybe . .. but ... even ... yes".

VII. Threats to Validity

There are two potential threats to validity. First, the dataset used in our study might not be representative of all English grammatical structures and word sentiments. We mitigate this threat by using widely used dataset in the NLP domain [22]. Second, using ranges of the sentiment scores in SST-2 to

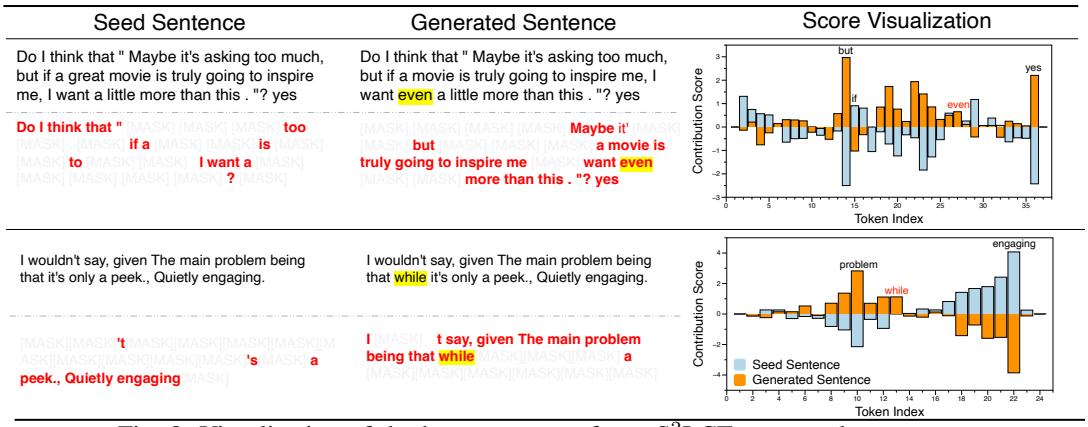

Fig. 8: Visualization of the buggy reason of two S^{2}LCT generated test cases.

determine the sentiment labels of search dataset may introduce incorrect labels. We performed a manual study to confirm that this is rare. [12, 13].

VIII. Related Work

NLP Testing. Apart from the accuracy-based testing scheme, recent works have considered model robustness as an aspect for evaluation. Belinkov and Bisk *et al.* [23] aimed to fail Neural Machine Translation (NMT) model by intentionally introducing noise in the input text. Pinjia *et al.* [24, 25] measured the robustness of NMT model by assuming syntactic and semantic relation between input and output. Ribeiro *et al.* [26] proposed an approach to generalize semantically equivalent adversarial rules. Rychalska *et al.* [27] measured drops in BLEU by corruption operation, and compared model robustness based on the amount of the drops. Iyyer *et al.* [28] introduced learning-based model for adversarial data augmentation.

In addition to the robustness on adversarial set, various other aspects of the NLP model have been considered for the robustness evaluation. Prabhakaran *et al.* [29] developed an evaluation framework to detect unintended societal bias in NLP models. Rottger *et al.* [4] introduced a functional test suite for hate speech detection in the NLP model. Ribeiro *et al.* [30] measured logical consistency of NLP model. However, in this work, we focused on the model evaluation over multiple perspectives and produced debugging information by comparing seed and expanded test cases.

Text Generation. Different data augmentation approaches have been proposed for text generation. One of methods in this category is easy data augmentation (EDA) [31]. It uses simple operations such as synonym replacement and random swap. Coulombe [32] implements rule-based transformations using regular expressions for text transformations such as insertion of spelling mistakes, data alterations, entity names, and abbreviations. Mixup interpolation, which was introduced in computer vision category, was implemented in text domain [33]. Learning-based language models has been used to generate text data for multiple tasks [34, 35]. While these approaches focus on obtaining synthetic training data, not testing NLP model over linguistic capabilities, our work generates text data for testing NLP model over linguistic capabilities.

Linguistic Capability Evaluation. Wang *et al.* [36, 37] proposed multiple diagnostic datasets to evaluate NLP models via natural language inference problems. More recently, CHECKLIST proposed an evaluation method of input-output behavior defined as linguistic capabilities. CHECKLIST generates behavior-guided inputs for validating the behaviors. [3]. Unlike prior work that relies on manual test case generation, we used structural information in text to generate test cases automatically.

NLP Model Debugging. Researchers have been explaining NLP model prediction and analyzing it for debugging the model. Ribeiro *et al.* [38] evaluated model prediction guided by human feedback providing relevance scores for words on the model prediction. Interactive error analysis [39] also has been proposed to evaluate model robustness. Zylberajch [40] used influence functions to generate model explanation, and it enables interactive debugging incorporating humans feedback explanation. Lertvittaya [41] proposed an approach to understand behavior of text classifier model and improve the model by disabling irrelevant hidden features. In this work, S^{2}LCT is useful for identifying the sources of model failure as shown in RQ4.

IX. Conclusions

In this paper, we present S^{2}LCT that automatically searches and generates seed test cases for each linguistic capability. S^{2}LCT expands the seeds into a sufficient number of test cases based on context-free grammar, covering diverse syntactic structures. We evaluated sentiment analysis models on the generated test cases. It is shown that S^{2}LCT ensures quality of test cases by showing consistency of test oracle with its input sentence and its linguistic capability. In addition, we showed that diversity in S^{2}LCT test cases leads to cover more region in sentiment analysis models, resulting in higher reliability on the testing. We also examined failure-inducing cases to understand the root causes of the bugs. These results demonstrate the correctness and usefulness S^{2}LCT for model evaluation.

TABLE IV: Results of BERT-base, da-ELECTRA and da-BERT-base hate speech detection models on S^2LCT test cases using all seeds.

Linguistic capability	**S^2LCT #Seeds**	**S^2LCT #Exps**	**S^2LCT #Fail**	**S^2LCT Fail rate[%]**	**S^2LCT #PassTo-Fail**
LC1: Expression of strong negative emotions (explicit)	140	799	BERT: 0 daELECTRA: 845 daBERT: 920	BERT: 0.00 daELECTRA: 89.99 daBERT: 97.98	BERT: 0 daELECTRA: 19 daBERT: 22
LC2: Description using very negative attributes (explicit)	140	2959	BERT: 0 daELECTRA: 3,011 daBERT: 3,067	BERT: 0.00 daELECTRA: 97.16 daBERT: 98.97	BERT: 0 daELECTRA: 30 daBERT: 23
LC3: Dehumanisation (explicit)	140	3124	BERT: 0 daELECTRA: 3,210 daBERT: 3,220	BERT: 0.00 daELECTRA: 98.35 daBERT: 98.65	BERT: 0 daELECTRA: 11 daBERT: 18
LC4: Implicit derogation	140	5664	BERT: 0 daELECTRA: 5,526 daBERT: 5,730	BERT: 0.00 daELECTRA: 95.21 daBERT: 98.73	BERT: 0 daELECTRA: 56 daBERT: 30
LC5: Direct threat	133	2689	BERT: 0 daELECTRA: 2,750 daBERT: 2,770	BERT: 0.00 daELECTRA: 97.45 daBERT: 98.16	BERT: 0 daELECTRA: 0 daBERT: 9
LC6: Threat as normative statement	140	4163	BERT: 0 daELECTRA: 4,198 daBERT: 4,261	BERT: 0.00 daELECTRA: 97.56 daBERT: 99.02	BERT: 0 daELECTRA: 12 daBERT: 1
LC7: Hate expressed using slur	805	17318	BERT: 0 daELECTRA: 17,212 daBERT: 17,597	BERT: 0.00 daELECTRA: 94.97 daBERT: 97.10	BERT: 0 daELECTRA: 81 daBERT: 70
LC8: Non-hateful use of slur	395	7881	BERT: 8,276 daELECTRA: 318 daBERT: 222	BERT: 100.00 daELECTRA: 3.84 daBERT: 2.68	BERT: 0 daELECTRA: 20 daBERT: 26
LC9: Hate expressed using profanity	1842	49743	BERT: 0 daELECTRA: 49,739 daBERT: 50,158	BERT: 0.00 daELECTRA: 96.42 daBERT: 97.23	BERT: 0 daELECTRA: 188 daBERT: 164
LC10: Non-Hateful use of profanity	701	15761	BERT: 16,462 daELECTRA: 541 daBERT: 549	BERT: 100.00 daELECTRA: 3.29 daBERT: 3.33	BERT: 0 daELECTRA: 51 daBERT: 58

TABLE V: Results of BERT-base, da-ELECTRA and da-BERT-base hate speech detection models on HateCheck test cases.

Linguistic capability	**S^2LCT #Seeds**	**S^2LCT #Fail**	**S^2LCT Fail rate[%]**
LC1: Derogation::Expression of strong negative emotions (explicit)	140	BERT: 0 daELECTRA: 134 daBERT: 136	BERT: 0.00 daELECTRA: 95.71 daBERT: 97.14
LC2: Derogation::Description using very negative attributes (explicit)	140	BERT: 0 daELECTRA: 136 daBERT: 138	BERT: 0.00 daELECTRA: 97.14 daBERT: 98.57
LC3: Derogation::Dehumanisation (explicit)	140	BERT: 0 daELECTRA: 138 daBERT: 138	BERT: 0.00 daELECTRA: 98.57 daBERT: 98.57
LC4: Derogation::Implicit derogation	140	BERT: 0 daELECTRA: 135 daBERT: 137	BERT: 0.00 daELECTRA: 96.43 daBERT: 97.86
LC5: Threatening language::Direct threat	133	BERT: 0 daELECTRA: 132 daBERT: 133	BERT: 0.00 daELECTRA: 99.25 daBERT: 100.00
LC6: Threatening language::Threat as normative statement	140	BERT: 0 daELECTRA: 136 daBERT: 139	BERT: 0.00 daELECTRA: 97.14 daBERT: 99.29
LC7: Slur usage::Hate expressed using slur	144	BERT: 0 daELECTRA: 142 daBERT: 144	BERT: 0.00 daELECTRA: 98.61 daBERT: 100.00
LC8: Slur usage::Non-hateful use of slur	111	BERT: 111 daELECTRA: 2 daBERT: 1	BERT: 100.00 daELECTRA: 1.80 daBERT: 0.90
LC9: Profanity usage::Hate expressed using profanity	140	BERT: 0 daELECTRA: 135 daBERT: 138	BERT: 0.00 daELECTRA: 96.43 daBERT: 98.57
LC10: Profanity usage::Non-Hateful use of profanity	100	BERT: 100 daELECTRA: 1 daBERT: 4	BERT: 100.00 daELECTRA: 1.00 daBERT: 4.00
LC11: Pronoun reference::Hate expressed through reference in subsequent clauses	140	BERT: 0 daELECTRA: 132 daBERT: 138	BERT: 0.00 daELECTRA: 94.29 daBERT: 98.57
LC12: Pronoun reference::Hate expressed through reference in subsequent sentences	133	BERT: 0 daELECTRA: 127 daBERT: 133	BERT: 0.00 daELECTRA: 95.49 daBERT: 100.00
LC13: Negation::Hate expressed using negated positive statement	140	BERT: 0 daELECTRA: 136 daBERT: 138	BERT: 0.00 daELECTRA: 97.14 daBERT: 98.57
LC14: Negation::Non-hate expressed using negated hateful statement	133	BERT: 133 daELECTRA: 5 daBERT: 2	BERT: 100.00 daELECTRA: 3.76 daBERT: 1.50
LC15: Phrasing::Hate phrased as a question	140	BERT: 0 daELECTRA: 137 daBERT: 139	BERT: 0.00 daELECTRA: 97.86 daBERT: 99.29
LC16: Phrasing::Hate phrased as a opinion	133	BERT: 0 daELECTRA: 128 daBERT: 131	BERT: 0.00 daELECTRA: 96.24 daBERT: 98.50
LC17: Non-hate grp. ident.::Neutral statements using protected group identifiers	126	BERT: 126 daELECTRA: 3 daBERT: 0	BERT: 100.00 daELECTRA: 2.38 daBERT: 0.00
LC18: Non-hate grp. ident.::Positive statements using protected group identifiers	189	BERT: 189 daELECTRA: 8 daBERT: 3	BERT: 100.00 daELECTRA: 4.23 daBERT: 1.59
LC19: Counter speech::Denouncements of hate that quote it	173	BERT: 173 daELECTRA: 4 daBERT: 4	BERT: 100.00 daELECTRA: 2.31 daBERT: 2.31
LC20: Counter speech::Denouncements of hate that make direct reference to it	141	BERT: 141 daELECTRA: 5 daBERT: 1	BERT: 100.00 daELECTRA: 3.55 daBERT: 0.71

REFERENCES

[1] M. Geva, Y. Goldberg, and J. Berant, "Are we modeling the task or the annotator? an investigation of annotator bias in natural language understanding datasets," *arXiv preprint arXiv:1908.07898*, 2019.

[2] S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. R. Bowman, and N. A. Smith, "Annotation artifacts in natural language inference data," *arXiv preprint arXiv:1803.02324*, 2018.

[3] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of nlp models with checklist," in *Association for Computational Linguistics (ACL)*, 2020.

[4] P. Röttger, B. Vidgen, D. Nguyen, Z. Waseem, H. Margetts, and J. B. Pierrehumbert, "Hatecheck: Functional tests for hate speech detection models," *arXiv preprint arXiv:2012.15606*, 2020.

[5] J.-t. Huang, J. Zhang, W. Wang, P. He, Y. Su, and M. R. Lyu, "Aeon: A method for automatic evaluation of nlp test cases," *arXiv preprint arXiv:2205.06439*, 2022.

[6] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[8] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[9] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *ArXiv*, vol. abs/1910.01108, 2019.

[10] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 2676–2686. [Online]. Available: https://www.aclweb.org/anthology/P18-1249

[11] N. Kitaev, S. Cao, and D. Klein, "Multilingual constituency parsing with self-attention and pre-training," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3499–3505. [Online]. Available: https://www.aclweb.org/anthology/P19-1340

[12] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, no. 2, p. 313–330, jun 1993.

[13] Sphinx and N. Theme. (2021) Nltk documentation. https://www.nltk.org/howto/corpus.html.

[14] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[15] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, 2013, pp. 1631–1642. [Online]. Available: https://aclanthology.org/D13-1170

[16] S. Baccianella, A. Esuli, and F. Sebastiani, "SentiWordNet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. Valletta, Malta: European Language Resources Association (ELRA), May 2010. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2010/pdf/769_Paper.pdf

[17] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu, "Texygen: A benchmarking platform for text generation models," *SIGIR*, 2018.

[18] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[19] S. Chen, S. Bateni, S. Grandhi, X. Li, C. Liu, and W. Yang, *DENAS: Automated Rule Generation by Knowledge Extraction from Neural Networks*. New York, NY, USA: Association for Computing Machinery, 2020, p. 813–825. [Online]. Available: https://doi.org/10.1145/3368089.3409733

[20] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 364–379.

[21] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[22] M. Husnain, M. M. S. Missen, N. Akhtar, M. Coustaty, S. Mumtaz, and V. Prasath, "A systematic study on the role of sentiwordnet in opinion mining," *Frontiers of Computer Science*, vol. 15, no. 4, pp. 1–19, 2021.

[23] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," *CoRR*, vol. abs/1711.02173, 2017. [Online]. Available: http://arxiv.org/abs/1711.02173

[24] P. He, C. Meister, and Z. Su, "Structure-invariant testing for machine translation," *CoRR*, vol. abs/1907.08710, 2019. [Online]. Available: http://arxiv.org/abs/1907.08710

[25] ——, "Testing machine translation via referential transparency," *CoRR*, vol. abs/2004.10361, 2020. [Online]. Available: https://arxiv.org/abs/2004.10361

[26] M. T. Ribeiro, S. Singh, and C. Guestrin, "Semantically equivalent adversarial rules for debugging NLP models," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 856–865. [Online]. Available: https://aclanthology.org/P18-1079

[27] B. Rychalska, D. Basaj, A. Gosiewska, and P. Biecek, "Models in the wild: On corruption robustness of neural nlp systems," in *Neural Information Processing*, T. Gedeon, K. W. Wong, and M. Lee, Eds. Cham: Springer International Publishing, 2019, pp. 235–247.

[28] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "Adversarial example generation with syntactically controlled paraphrase networks," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana:

Association for Computational Linguistics, Jun. 2018, pp. 1875–1885. [Online]. Available: https://aclanthology.org/N18-1170

[29] V. Prabhakaran, B. Hutchinson, and M. Mitchell, "Perturbation sensitivity analysis to detect unintended model biases," *CoRR*, vol. abs/1910.04210, 2019. [Online]. Available: http://arxiv.org/abs/1910.04210

[30] M. T. Ribeiro, C. Guestrin, and S. Singh, "Are red roses red? evaluating consistency of question-answering models," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 6174–6184. [Online]. Available: https://aclanthology.org/P19-1621

[31] J. Wei and K. Zou, "EDA: Easy data augmentation techniques for boosting performance on text classification tasks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 6382–6388. [Online]. Available: https://aclanthology.org/D19-1670

[32] C. Coulombe, "Text data augmentation made simple by leveraging NLP cloud apis," *CoRR*, vol. abs/1812.04718, 2018. [Online]. Available: http://arxiv.org/abs/1812.04718

[33] H. Guo, Y. Mao, and R. Zhang, "Augmenting data with mixup for sentence classification: An empirical study," *CoRR*, vol. abs/1905.08941, 2019. [Online]. Available: http://arxiv.org/abs/1905.08941

[34] Q. Xie, Z. Dai, E. H. Hovy, M. Luong, and Q. V. Le, "Unsupervised data augmentation," *CoRR*, vol. abs/1904.12848, 2019. [Online]. Available: http://arxiv.org/abs/1904.12848

[35] A. Anaby-Tavor, B. Carmeli, E. Goldbraich, A. Kantor, G. Kour, S. Shlomov, N. Tepper, and N. Zwerdling, "Not enough data? deep learning to the rescue!" *CoRR*, vol. abs/1911.03118, 2019. [Online]. Available: http://arxiv.org/abs/1911.03118

[36] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *CoRR*, vol. abs/1804.07461, 2018. [Online]. Available: http://arxiv.org/abs/1804.07461

[37] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *CoRR*, vol. abs/1905.00537, 2019. [Online]. Available: http://arxiv.org/abs/1905.00537

[38] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," *CoRR*, vol. abs/1602.04938, 2016. [Online]. Available: http://arxiv.org/abs/1602.04938

[39] T. Wu, M. T. Ribeiro, J. Heer, and D. Weld, "Errudite: Scalable, reproducible, and testable error analysis," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 747–763. [Online]. Available: https://aclanthology.org/P19-1073

[40] H. Zylberajch, P. Lertvittayakumjorn, and F. Toni, "Hildif: Interactive debugging of nli models using influence functions," *Proceedings of the First Workshop on Interactive Learning for Natural Language Processing*, 2021.

[41] P. Lertvittayakumjorn, L. Specia, and F. Toni, "Find: Human-in-the-loop debugging deep text classifiers," 2020. [Online]. Available: https://arxiv.org/abs/2010.04987