Automated Testing Linguistic Capabilities of NLP Models

Anonymous Author(s)

Abstract

Natural language processing (NLP) has gained widespread adoption in the development of diverse real-world applications. However, the black-box nature of the backend neural networks in NLP applications poses a challenge when evaluating their performance, let alone ensuring it. Recent research has proposed testing techniques to enhance the trustworthiness of NLP-based applications. However, existing works either generate invalid test cases (i.e. those with incorrect grammar) or rely on manually designed test cases, which may limit their diversity. To address this limitation, we propose S^{2}LCT, an automated testing technique for validating NLP applications based on their linguistic capabilities. S^{2}LCT first transforms test cases from existing real-world datasets into seed inputs that belong to individual linguistic capabilities. S^{2}LCT then generates a sufficient number of test cases by expanding syntax of the seed inputs based on their context-free grammar (CFG). In the syntax expansion process, S^{2}LCT validates that the expanded test cases maintain the same semantic and capability as the seed inputs, thereby automatically constructing a test oracle and linguistic capability for the expanded test cases. We evaluate S^{2}LCT on two widely adopted NLP tasks, sentiment analysis and hate speech detection, to determine its performance in terms of diversity, effectiveness, and consistency. The results showed that S^{2}LCT generates test cases that are at least 88% more diverse than those produced by state-of-the-art techniques, as measured by Self-BLEU and production rule coverage metrics. Additionally, S^{2}LCT was found to produce a larger number of NLP model failures.

ACM Reference Format:
Anonymous Author(s). 2023. Automated Testing Linguistic Capabilities of NLP Models. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 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 model behavior on fine-grained aspects of NLP models. To address this limitation of the traditional quality metric, an increasing number of evaluation methods has been introduced to evaluate an NLP model in different aspects such as robustness on adversarial examples [1, 34, 42, 60, 63] and bias on demographic groups [2, 38, 55, 59]. However, such metrics still assess specific facet of the NLP models, and they still fail to validate how well a model behaves in different linguistic scenarios [16, 19].

Recent research has proposed methods to assess an NLP model on a set of linguistic capabilities in different NLP applications [50, 51]. A linguistic capability is a specific linguistic scenario between input and output observed in the domain of NLP application. Compared with the traditional evaluation metrics, linguistic capability-based testing avoids the overestimation of the model performance by measuring variances of model performance over the capabilities. Thus, it is able to give detail assessment on strength and weakness of a given NLP model. Nevertheless, existing linguistic capability-based testing approaches are limited to facilitate automated and comprehensive testing. They rely on manually designed word substitution-based templates. The test cases are generated by replacing words on the place holders in the templates [50, 51] Therefore, their test cases do not cover diverse sentence structures, preventing comprehensive testing of NLP model on linguistic capabilities.

To address this issue, we present S^{2}LCT, an automated linguistic capability-based framework for testing NLP models. The goal of this work is to generate a diverse linguistic capability-based test suite automatically. To achieve the goal, S^{2}LCT needs to address two requirements: *(i) Capability-based Generation.* 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 text generation. Whether a test sentence is suitable for evaluating a specific linguistic capability is decided by its relevancy to this linguistic capability. It is challenging to maintain relevancy between a test sentence and its linguistic capability when generating the test sentence. This is because the linguistic capability is defined on a specific mixture of syntax and semantics of the sentences. 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. Existing metamorphic or adversarial testing [1, 34, 42, 60, 63] considers only labels of generated test cases without linguistic capacity guarantee. To address this difficulty, S^{2}LCT defines *specifications* (search-based and enumerative seed generation non-terminals) from linguistic capabilities, and generates seed test cases by perturbing test cases into the non-terminals. In addition, S^{2}LCT analyzes the parse trees of the seed sentences to identify possible expansion and validates coherence of linguistic capability relevancy between each seed and its expansion.

ICSE 2024, April 2024, Lisbon, Portugal
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Automated Test Oracle. For NLP model testing, test oracle is usually determined by understanding the semantics of texts, which requires domain knowledge for different NLP tasks [4, 18, 20, 56]. Thus, the current testing practice requires manual effort to create the test oracles of the holdout data, which is time-consuming. 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 testing [18, 20, 24, 56]. Several 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, 36, 44, 58, 62, 66]. 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. We address the challenge by only accepting validated test oracles defined by input-output relations determined by the linguistic capability specification.

In this work, as a first step, we consider sentiment analysis and hate speech detection as the NLP applications for automated linguistic capability-based testing. We demonstrate the effectiveness of S^2LCT by evaluating three sentiment analysis and two hate speech detection models.

In summary, we made the following contributions in this work:

- We design and implement an automated linguistic capability-based testing approach, S^2LCT, capable of producing diverse test cases while performing capability-based text generation using automated test oracle.
- We evaluate text classification models on test cases generated by S^2LCT on 10 and 14 linguistic capabilities for sentiment analysis and hate speech detection, respectively. Comparing with the state-of-the-art linguistic capability-based testing baselines, we find that S^2LCT produces at least 88% more diverse test cases, measured in Self-BLEU and production rule coverage, and a larger number of NLP model failures in 21 out of 24 linguistic capabilities over the two NLP applications. We also analyze the root causes of the misclassifications, and find that seeds and expanded test cases produced by S^2LCT are useful in helping developers understand bugs in the model.
- We perform a manual study and confirm that S^2LCT test cases are accurately labelled and categorized into linguistic capabilities.

2 Background & Motivation

Linguistic capability-based testing introduces task-dependent linguistic capabilities for monitoring model performance on each linguistic capability [50, 53]. Test cases that conform to the linguistic capability are generated, and accuracy of the model prediction on them is measured. We use CHECKLIST [50], a state-of-the-art linguistic capability-based testing approach to illustrate the idea. In Figure 1, templates defined in lines 4 to 9 are used for evaluating the sentiment analysis model on the linguistic capability "Sentiment change over time, present should prevail". These templates contain placeholders, *pos_adj*, *neg_adj*, and *change*. Values for the placeholders are a collection of words defined in lines 1 to 3. For each template, CHECKLIST fills in all the combinations of the possible

```
1 pos_adj = ['good', 'great', 'excellent', 'amazing', ...]
2 neg_adj = ['awful', 'bad', 'horrible', 'weird', ...]
3 change = ['but', 'even though', 'although']
4 t = editor.template([
5     'I used to think this airline was {neg_adj}, {change} now I think
          it is {pos_adj}.',
6     'I think this airline is {pos_adj}, {change} I used to think it was {
          neg_adj}.',
7     'In the past I thought this airline was {neg_adj}, {change} now I
          think it is {pos_adj}.',
8     'I think this airline is {pos_adj}, {change} in the past I thought it
          was {neg_adj}.'] ,
9 change=change, ... ,labels=2)
```

Figure 1: Example of CHECKLIST template

values of placeholders to generate sentences under this linguistic capability. For example, sentences such as "I used to think this airline was bad, but now I think it is good." and "In the past I thought this airline was weird, although now I think it is great." are generated. In this manner, all test cases generated from the template conform to the linguistic capability of interest, i.e., sentiment change over time for this example. These test cases can be used to assess how well a sentiment analysis model understands sentiment change over time. Therefore, linguistic capability testing is beneficial to evaluate not only how model accurately predicts, but also how model understands the metamorphic relation between input and output and if it behaves as the linguistic capability specifies.

Despite the potential usefulness of linguistic capability-based testing, all existing capability testing work [50, 51] shares common limitations. First, they require manual work for defining templates and its placeholder values for every linguistic capabilities, as illustrated in Figure 1. This manual work is costly, and it makes the test case generation rely on relatively simple template structure, thus limiting test coverage on each linguistic capability. Moreover, performing word substitution for the template placeholders produces similar test cases with regard to input text and structure. The limited diversity in test cases results in bias in model evaluation on the linguistic capability with the test cases. These limitations motivated the design of our approach.

3 Specification- and Syntax-based Linguistic Capability Testing

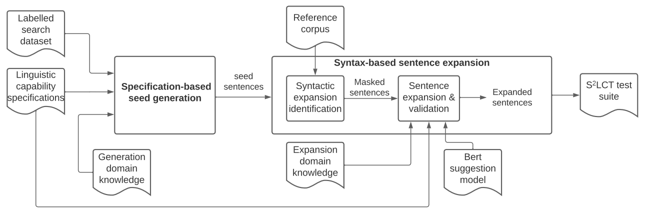

Labelled search dataset
Linguistic capability specifications
Generation domain knowledge
Specification-based seed generation
seed sentences
Reference corpus
Syntax-based sentence expansion
Syntactic expansion identification
Masked sentences
Sentence expansion & validation
Expanded sentences
Expansion domain knowledge
Bert suggestion model
S2LCT test suite

Figure 2: Overview of S^2LCT.

We design and implement a new NLP model testing framework, S^2LCT, which automatically generates test cases with oracles for defined linguistic capabilities 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 generates the initial set of seed test cases based on generation rules. Generation rule is consisted of operations and non-terminals. S^{2}LCT provides two built-in operations `Combine` and `Replace` and two types of non-terminals, search-based and enumerative non-terminals (Section 3.1). To increase the diversity of test cases generated by S^{2}LCT, we utilize a *syntax-based sentence expansion* phase (Section 3.2). This phase begins by performing a syntax analysis to identify part-of-speech (PoS) tags that can be added to the seed test cases, by comparing the PoS parse trees of the seed test cases with a large reference corpus of sentences. The identified tags are then inserted into the seed test cases as a *mask*. A masked language model, such as BERT [9], is then used to suggest words that can fill in the mask. Finally, the resulting sentence is checked to ensure it is consistent with the seed's label and linguistic capability. The generated test suite includes both the original seed test cases and the expanded test cases. This approach enables S^{2}LCT to cover a wider range of syntactic structures, enhancing its effectiveness in evaluating NLP models.

3.1 Specification-based Seed Generation

S^{2}LCT generates the initial set of seed test cases based on generation rules. Each generation rule is consisted of operations and non-terminals. S^{2}LCT provides two built-in operations `combine` and `replace`. The `combine` function takes non-terminals as parameters, perturbs them in the order as they appear, and returns a collection of seed sentences. The order of the parameters defines the scope of the linguistic capability. Specifically, S^{2}LCT supports two types of non-terminals: enumerative and search-based non-terminals. S^{2}LCT supports four types of *enumerative* non-terminals, *prefix*, *sent*, *infix* and *postfix*. A *sent* non-terminal is a collection of the sentences returned by applying the search-based predicates on the real-world dataset (i.e., search-based non-terminals). *prefix*, *infix*, and *postfix* non-terminals are sets of user-defined clauses, defined by their relative positions to the *sent* non-terminal.

In addition to enumerative non-terminals, S^{2}LCT also supports search-based non-terminals. Search-based non-terminals allow users to define search predicates to construct constraints (in the form of combinational logic) for each linguistic capability. Search-based non-terminals enable S^{2}LCT to find candidate sentences in given real-world datasets matching (partial) description of each linguistic capability. S^{2}LCT supports two types of search predicates as boolean-valued functions, *implicit* and *explicit*, which evaluate and verify that a test case meets the criteria on explicit and implicit linguistic attributes, respectively. An explicit attribute is an attribute that can be verified directly in the search dataset, while an implicit attribute requires external domain knowledge. S^{2}LCT provides a built-in predicate for each type of the attributes (`hasattr` for explicit attribute and `contain` for implicit attribute). For example, for a sentiment analysis, a predicate for any test cases labeled with negative sentiment can be expressed as `hasattr`$(label, negative)$. For finding test cases with any neutral adjective words, the predicate can be expressed as $contain(pos, neutral\_adj)$. The difference between the two cases is that the latter case requires the analysis of each word in the sentence to identify its semantic and tag of PoS, which may not be obtained directly in the search dataset. Thus, external linguistic knowledge of tag of PoS and sentiment of word is needed.

In later phase, after a whole test case is generated, we use combinational logic of user-defined predicates to evaluate if the test case is within the scope of a linguistic capability. For example, the second row in Table 1 shows the search predicates of the linguistic capability "*short sentences with sentiment-laden adjectives*". The search predicates define that the sentences with less than 10 tokens (explicit attribute) and have positive/negative labels (explicit attribute) containing positive/negative adjectives or nouns (implicit attributes) are within this linguistic capability.

The third row in Table 1 shows an example of the search-based and enumerative predicates of the linguistic capability "*negation of negative at the end, should be positive or neutral*" for sentiment analysis. The search-based predicate only finds negative test case using the *hasattr*, expressed as $hasattr(label, negative)$. In its enumerative predicates, *sent* maps to the searched negative test cases, and *prefix* and *postfix* are user-defined clauses appearing before and after *sent*, respectively.

In addition to the `combine` operation, S^{2}LCT also supports `replace` operations that allow linguistic attributes of non-terminals be changed in specific ways. In this work, we employ negation for some linguistic capabilities that requires negation of sentences. Our complete set of seed generation rules is available on the project website.[1]

Running example. We use the running example shown in Figure 3 to illustrate how S^{2}LCT works. The first column illustrates the specification-based seed generation phase. Given the linguistic capability "short sentences with neutral adjectives and nouns" for the sentiment analysis task, a sentence "Or both." is selected as a seed for the linguistic capability as it satisfies the search-based predicates which require the short sentence be labeled as neutral and has neutral adjective (i.e. both). Other sentences in the first column are excluded because their labels are not neutral and/or they include adjectives/nouns that are not neutral, violating the search-based predicates of this linguistic capability.

3.2 Syntax-based Sentence Expansion

The seed test cases generated may still be limited in their syntactic structures because the search dataset may not cover all the syntax. To address this limitation, we design the syntax-based sentence expansion phase to extend the seed sentences to cover diverse syntactic structures while still conform to its respective linguistic capability specification. Our insight is that sentences commonly used in real life cover diverse and realistic syntactic structures that can be used as the basis for the expansion. Thus, in this stage, we differentiate the parse trees between the seed sentences and a set of reference sentences obtained from a large real-world dataset. The extra tags of PoS 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.

3.2.1 *Syntax Expansion Identification*

Algorithm 1 shows how masks are identified for the seed sentences. It takes the parse trees of the

[1] https://sites.google.com/view/s2lct/home

Table 1: Search-based and enumerate predicates for two linguistic capabilities of sentiment analysis task.

Linguistic capability	**Search-based and enumerate predicates** Shiyi: Generation rule?
Short sentences with sentiment-laden adjectives	*Seed* ←**hasattr**(length,<10)∧((**contain** (tag_of_pos,positive_adj)∨**contain** (tag_of_pos,positive_noun) ∧**hasattr**(label,positive))∨(**contain** (tag_of_pos,negative_adj)∨**contain**(tag_of_pos,negative_noun)∧**hasattr**(label,negative)))
Negated negative should be positive or neutral	*Seed* ←**replace**(*S*, "is", ["is not","isn't"])+**replace**(*S*, "are", ["are not","aren't"]) *S* ←**hasattr**(clause,'(This|That|These|Those) (is|are)')∧**hasattr**(label,negative)

Negation of negative at the end, should be positive or neutral	*Seed* ←**combine**(prefix,S_{neg},postfix)
	prefix:="I agreed that"|"I thought that"
	S_{neg} ←**hasattr**(label,negative)
	postfix:="but it wasn't"|"but I didn't"

Algorithm 1 Syntax expansion identification algorithm.

```
 1: Input: Parse trees of seed sentences $S$, reference CFG $R$
 2: Output: Set of masked sentences $M$
 3: for each part tree $s$ from $S$ do
 4:     for each production $s\_prod$ from $s$ do
 5:         $s\_lhs = s\_prod.lhs$
 6:         $s\_rhs = s\_prod.rhs$
 7:         for each $r\_rhs$ from $R[s\_lhs]$ do
 8:             if $s\_rhs \subset r\_rhs$ then
 9:                 $M = M \cup insertMask(\text{r\_rhs-s\_rhs}, s)$
10: return $M$
```

seeds, generated by the Berkeley Neural Parser [29, 30], and a reference context-free grammar (CFG) (i.e., the reference dataset in Figure 2) as inputs. Overall, this algorithm identifies the discrepancy between the seed syntax and the reference grammar, which is representative of the distribution of real-world language usage, to decide how a seed and what syntax in the seed can be expanded, producing a set of masked sentences.

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 CFG 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. All the masked sentences of each seed are returned at line 14.

Running example. The second and third columns in Figure 3 illustrate how Algorithm 1 is used to generate a masked sentence. The second column shows the parse tree of the seed sentence "Or both.", which 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 the reference CFG 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 third column. The masked sentence "Or both $\{MASK\}$." is the result of the left-to-right traversal of this leaves of the expanded parse tree.

3.2.2 Sentence Expansion and Validation To expand a masked sentence, the words to fill in the masks are suggested by the BERT model [9]. BERT is a transformer-based natural language model that is pre-trained on masked token prediction task. 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 all suggested words may no longer satisfy the linguistic capability specification or the expanded grammar symbol. Therefore, we perform *validation* on the suggested words and only accept them if the following three criteria are met.

First, the tag of PoS 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". In our implementation, we use the SpaCy [23] library to extract the tag of PoS for each suggested word.

Second, we require the suggested words be neutral. In general, changing even just one word may change the overall label and/or linguistic capability of a sentence, which violates the goal of S^{2}LCT. To reduce this risk, we only accept neutral words from the suggested words, which requires us to use the expansion domain knowledge to verify the sentiment of each suggested word.

Third, we verify that the expanded sentences satisfy the same linguistic capability specification as their seed sentences. An expanded sentence may no longer be within the scope of its seed's linguistic capability. For example, the predicate shown in the second row of Table 1, ***hasattr****(length,<10)*, may no longer hold after expanding a seed sentence with multiple words. We only accept an expanded sentence if the search-based predicates are still satisfied and the expansion does not happen within an enuramerate non-terminal.

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

4 Experimental Setup

In this section, we present the setup of our experiments. We answer the following research questions (RQs):

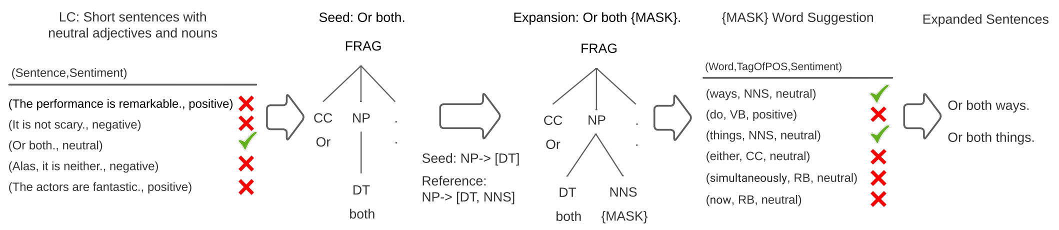

LC: Short sentences with neutral adjectives and nouns
(Sentence,Sentiment)
(The performance is remarkable., positive)
(It is not scary., negative)
(Or both., neutral)
(Alas, it is neither., negative)
(The actors are fantastic., positive)
Seed: Or both.
FRAG
CC NP .
Or
DT
both
Seed: NP-> [DT]
Reference: NP-> [DT, NNS]
Expansion: Or both {MASK}.
FRAG
CC NP .
Or
DT NNS
both {MASK}
{MASK} Word Suggestion
(Word,TagOfPOS,Sentiment)
(ways, NNS, neutral)
(do, VB, positive)
(things, NNS, neutral)
(either, CC, neutral)
(simultaneously, RB, neutral)
(now, RB, neutral)
Expanded Sentences
Or both ways.
Or both things.

Figure 3: Running example of S^2LCT.

- **RQ1 Diversity.** Can S^2LCT generate more diverse test cases than existing approaches?
- **RQ2 Effectiveness.** Is S^2LCT more effective than existing approaches at generating test cases that can trigger more errors in the model?
- **RQ3 Consistency.** Can S^2LCT maintain consistency in terms of labels, linguistic capabilities, and semantics throughout the expansion process?

Table 2: The NLP model used in our evaluation

Tasks	Model Name	API URL	#Downloads
Semantic Analysis	BERT-base	bert-base-uncased-SST-2	109,143
Semantic Analysis	RoBERTa-base	roberta-base-SST-2	6,099
Semantic Analysis	DistilBERT-base	distilbert-base-uncased-SST-2	31
Hate Speech Detection	dehate-BERT	dehatebert-mono-english	1,366
Hate Speech Detection	twitter-RoBERTa	twitter-roberta-base-hate	40,097

4.1 Experimental Subjects

NLP Models. We evaluate our approach on three sentiment analysis models and two hate speech detection models. We obtain our evaluation models from the HuggingFace model hub [25]. Table 2 presents the models and their corresponding API URLs. The "API URL" column displays the public URL of each model, while the "# of downloads" column indicates the number of downloads for each model in Jan. 2023. Based on the information provided in Table 2, it is evident that all models utilized in our evaluation have been widely adopted in real-world settings, with a considerable number of downloads. For readers who seek more comprehensive information about each model, we recommend referring to the corresponding API URL of each model.

Datasets. In our evaluation of sentiment analysis models, we utilize SST [54] as our initial labeled search dataset. Meanwhile, for assessing models designed for hate speech detection, we make use of HateXplain [40] as our initial dataset. SST is a corpus of movie reviews that consists of 11,855 sentences, each of which has been labeled as either negative, neutral, or positive to indicate the expressed sentiment in the sentence. HateXplain is a dataset that has been collected from social media platforms Twitter and Gab. It consists of 20,148 sentences, with 9,055 of them being from Twitter and 11,093 from Gab. Each sentence in this dataset has been labeled as either "hate" or "non-hate" to indicate the presence or absence of hate speech in the sentence [40].

Baselines. For RQ1, we first compare S^2LCT against CHECKLIST [50] and Hatecheck [51]. After that, we also compare S^2LCT against MT-NLP, Alzantot-attack, BERT-Attack and SememePSO-attack [1, 34, 38, 63]. For RQ2, we compare S^2LCT against CHECKLIST and Hatecheck again.

$$Syntactic\ Diversity(\mathcal{X}) = ||\{\mathcal{P}(x) \mid \forall x \in \mathcal{X}\}|| \qquad (1)$$

4.2 Evaluation Metrics

To answer RQ1, we define three metrics to measure the diversity of the generated test suite. Our first metric is *Self-BLEU*, which was introduced by [68]. Self-BLEU is defined as the average BLEU score [43] over all reference sentences, ranging between 0 and 1. A higher Self-BLEU score indicates lower diversity in the test suite, while a lower score indicates greater diversity. However, since Self-BLEU cannot represent the syntactic diversity of a test suite, we have proposed a new metric, *syntactic diversity*, to evaluate the diversity of the test suite. The syntactic diversity of a test suite $\mathcal{X}$ is defined as the number of production rules covered in this test suite. The formal definition of syntactic diversity is shown in (1), where $\mathcal{P}$ is the Berkeley Neural Parsing function [29, 30] that return the production rule of the given sentence. Our final metric is *neuron coverage*. We follow the approach presented by Ma et al. [36], 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. [36], boundary coverage (BoundCov) and strong activation coverage (SActCov), to evaluate the test suite diversity.

$$\begin{aligned} UpperCorner(\mathcal{X}) &= \{n \in N | \exists x \in \mathcal{X} : f_n(x) \in (high_n, +\infty)\}; \\ LowerCorner(\mathcal{X}) &= \{n \in N | \exists x \in \mathcal{X} : f_n(x) \in (-\infty, low_n)\}; \end{aligned} \qquad (2)$$

Equation 2 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' upper and lower output bounds on the model training dataset respectively. Equation 2 can be interpreted as the collection of neurons that emit outputs beyond the model's numerical boundary.

$$\begin{aligned} BoundCov(\mathcal{X}) &= \frac{|UpperCorner(\mathcal{X})| + |LowerCorner(\mathcal{X})|}{2 \times |N|} \\ SActCov(\mathcal{X}) &= \frac{|UpperCorner(\mathcal{X})|}{|N|} \end{aligned} \qquad (3)$$

The definition of our model coverage metrics is shown in Equation 3, 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 diversity.

For RQ2, our goal is to answer whether S^{2}LCT is more effective than other methods for generating test cases that can trigger incorrect predictions. Thus, we measure three key metrics: (1) the number of test cases generated, (2) the number of failed test cases, and (3) the failure rates of the generated test cases. Additionally, we also report on the number of expanded test cases that failed but whose corresponding seed test cases passed (Pass-to-Fail) using S^{2}LCT.

$$\begin{aligned}
LabelCons &= \frac{1}{\#Sample} \cdot \sum_i \delta(label_{S^2LCT} = label_{human}) \\
LCRel_{AVG} &= \frac{1}{\#Sample} \cdot \sum_i Norm(LCRel_i) \qquad (4) \\
ExpValidity_{AVG} &= \frac{1}{\#ExpSample} \cdot \sum_i Norm(ExpValidity_i)
\end{aligned}$$

To answer RQ3, we introduce three new metrics: the label consistent rate (*LabelCons*), the linguistic capability consistent rate ($LCRel_{AVG}$), and the semantic consistent rate ($ExpValidity_{AVG}$). The formal definitions of these metrics are listed in Equation 4.

4.3 Experimental Process

RQ1 Process. In the evaluation, we collected 200, 400, 600, 800 and 1000 test cases for sentiment analysis and 10000, 50000, 100000, 150000 and 200000 test cases for hate speech detection, randomly selected S^{2}LCT seed and expanded sentences. We then computed the median of Self-BLEU and production rule coverage scores over all linguistic capabilities. We repeated this computation with different S^{2}LCT seeds over 5 trials and reported the median. We also evaluated S^{2}LCT expansion phase by generating expanded sentences from CHECKLIST and Hatecheck as seeds. We collected up to 200 randomly selected test cases from CHECKLIST and Hatecheck and generate their expanded sentences. We computed the median of Self-BLEU and production rule coverage scores from the sentences over all linguistic capabilities. We repeated the computation with different S^{2}LCT seeds over 3 trials and reported the median over the 3 trials. In addition, we compared Self-BLEU and production rule coverage scores between S^{2}LCT and text generation baselines. First, we generate two groups of sentences from 100 randomly selected S^{2}LCT seeds for each sentiment analysis and hate speech detection using S^{2}LCT expansion and syntax-based text generation baseline. Self-BLEU and production rule coverage scores of the two groups of sentences were then compared. Second. we generate two groups of sentences from 50 randomly selected S^{2}LCT seeds for sentiment analysis using S^{2}LCT expansion and adversarial text generation baselines. Likewise, we compared Self-BLEU and production rule coverage scores of the two groups of sentences.

For the model coverage metric, we begin by feeding the training dataset of each NLP model under test into the system in order to compute the lower and upper bounds for each neuron. Then, we select an equal number of test cases from both S^{2}LCT and CHECKLIST to construct the test suite and calculate the corresponding model coverage metrics.

RQ2 Process. We address RQ2 by evaluating 5 models in Table 2 on test cases of S^{2}LCT and linguistic capability-based testing baselines for sentiment analysis and hate speech detection. For each linguistic capability, we measure the number of test cases generated from the baselines, S^{2}LCT seeds and their expansions. We calculate the number of failures and fail rate of the 5 models. In addition, we compare model performances on test cases between S^{2}LCT seeds and their expansions, and measure the number of Pass-to-Fail cases.

RQ3 Process. To address RQ3, we conduct a manual study to evaluate the three consistency metrics listed in Equation (4) for the test suite generated by S^{2}LCT. For each task, we randomly sampled 100 S^{2}LCT seed sentences. We divide these seeds to two sets (i.e., 50 sentences in each set). For each sampled seed sentence, we randomly obtain one of its expanded sentences. This forms the two sets of sentences, each with 50 seeds and 50 corresponding expanded sentences. We recruited two participants for each task; all of them are graduate students with no knowledge about this work. Each of them was assigned a different set of sentences, and asked to provide three 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 to 5 representing "strongly negative" to "strongly positive" for sentiment analysis and "strongly normal" to "strongly hateful" for hate speech detection, respectively. *(3) Validity score of expanded sentence*: this score measures the validity of the use of label of a seed sentence for its associated S^{2}LCT expanded sentence. The scores are discrete ranging from 1 ("strongly not consistent") to ("strongly consistent").

Implementation Details. In our implementation, we obtained our reference CFG from the Penn Treebank corpus [39]. Additionally, we utilized SentiWordNet [3], which is a lexical sentiment resource, as the domain-specific knowledge for sentence expansion. All experiments were conducted on a Ubuntu 14.04 server with three Intel Xeon E5-2660 v3 CPUs @2.60GHz, eight Nvidia 1080Ti GPUs, and 500GB of RAM.

5 Experimental Results

This section presents the experimental results and the answers to the RQs. More results are available at the S^{2}LCT repository.[2]

5.1 RQ1: Diversity

Our results show that *S^{2}LCT produced test suites with significantly more diversity than baselines did.*

Self-BLEU and Production Rule Coverage. Figure 4 compares the Self-BLEU and production rule coverage scores of the test suite generated by S^{2}LCT in contrast to those of CHECKLIST and Hatecheck. The x-axis shows the sizes of the generated test suite, and the y-axis shows the metric scores. The figure displays the median

[2]https://github.com/csresearcher27/s2lct_project

Table 3: Results of BERT-base, RoBERTa-base and DistilBERT-base sentiment analysis models on S^{2}LCT test cases using all seeds. BERT-base, RoBERTa-base and DistilBERT-base models are denoted as BERT, RoBERTa and dstBERT, respectively.

Linguistic capability	Cklst #TCs	S^{2}LCT #Seeds	S^{2}LCT #Exps	S^{2}LCT/Cklst #Fail	S^{2}LCT/Cklst Fail rate[%]	S^{2}LCT #PassToFail
LC1: Short sentences with neutral adjectives and nouns	1,716	19	51	BERT: 60/1,330	BERT: 85.71/77.51	BERT: 9
				RoBERTa: 55/1,391	RoBERTa: 78.57/81.06	RoBERTa: 2
				dstBERT: 68/1,661	dstBERT: 97.14/96.79	dstBERT: 0
LC2: Short sentences with sentiment-laden adjectives	8,658	160	262	BERT: 25/26	BERT: 5.92/0.30	BERT: 5
				RoBERTa: 39/139	RoBERTa: 9.24/1.61	RoBERTa: 14
				dstBERT: 18/125	dstBERT: 4.27/1.44	dstBERT: 10
LC3: Sentiment change over time, present should prevail	8,000	75,159	343,214	BERT: 99,312/1,680	BERT: 23.74/21.00	BERT: 10,357
				RoBERTa: 208,313/829	RoBERTa: 49.79/10.36	RoBERTa: 11,472
				dstBERT: 262,994/2,532	dstBERT: 62.86/31.65	dstBERT: 9,808
LC4: Negated negative should be positive or neutral	6,786	67	503	BERT: 523/799	BERT: 91.75/11.77	BERT: 20
				RoBERTa: 498/218	RoBERTa: 87.37/3.21	RoBERTa: 9
				dstBERT: 494/734	dstBERT: 86.67/10.82	dstBERT: 6
LC5: Negated neutral should still be neutral	2,496	26	194	BERT: 207/2,427	BERT: 94.09/97.24	BERT: 11
				RoBERTa: 204/2,304	RoBERTa: 92.73/92.31	RoBERTa: 6
				dstBERT: 213/2,450	dstBERT: 96.82/98.16	dstBERT: 10
LC6: Negation of negative at the end, should be positive or neutral	2,124	18,576	97,897	BERT: 116,049/1,871	BERT: 99.64/88.09	BERT: 67
				RoBERTa: 115,676/445	RoBERTa: 99.32/20.95	RoBERTa: 90
				dstBERT: 114,556/2,124	dstBERT: 98.35/100.00	dstBERT: 325
LC7: Negated positive with neutral content in the middle	1,000	24,328	184,328	BERT: 189,935/860	BERT: 91.03/86.00	BERT: 1,972
				RoBERTa: 153,686/416	RoBERTa: 73.66/41.60	RoBERTa: 7,007
				dstBERT: 175,323/865	dstBERT: 84.02/86.50	dstBERT: 5,003
LC8: Author sentiment is more important than of others	8,528	68,284	465,291	BERT: 152,009/3,741	BERT: 28.49/43.87	BERT: 8,878
				RoBERTa: 105,152/2,693	RoBERTa: 19.71/31.58	RoBERTa: 8,487
				dstBERT: 162,426/3,535	dstBERT: 30.44/41.45	dstBERT: 12,729
LC9: Parsing sentiment in (question, yes) form	7,644	15,465	102,203	BERT: 7,097/253	BERT: 6.03/3.31	BERT: 1,590
				RoBERTa: 6,226/32	RoBERTa: 5.29/0.42	RoBERTa: 1,489
				dstBERT: 5,470/52	dstBERT: 4.65/0.68	dstBERT: 1,151
LC10: Parsing sentiment in (question, no) form	7,644	15,483	102,214	BERT: 89,155/4,056	BERT: 75.75/53.06	BERT: 1,722
				RoBERTa: 100,351/4,576	RoBERTa: 85.26/59.86	RoBERTa: 1,452
				dstBERT: 111,874/6,440	dstBERT: 95.05/84.25	dstBERT: 575

Table 4: Results of dehate-BERT and tweeter-RoBERTa hate speech detection models on S^{2}LCT test cases using all seeds. dehate-BERT and tweeter-RoBERTa are denoted as BERT, RoBERTa respectively.

Linguistic capability	Htck #TCs	S^{2}LCT #Seeds	S^{2}LCT #Exps	S^{2}LCT/Htck #Fail	S^{2}LCT/Htck Fail rate[%]	S^{2}LCT #PassToFail
LC1: Hate expressed using slur	144	203	1,171	alex-BERT: 435/108	alex-BERT: 31.66/75.00	alex-BERT: 16
				CNERG-BERT: 26/56	CNERG-BERT: 1.89/38.89	CNERG-BERT: 12
LC2: Non-hateful use of slur	111	997	4,422	alex-BERT: 3,835/18	alex-BERT: 70.77/16.22	alex-BERT: 29
				CNERG-BERT: 4,484/68	CNERG-BERT: 82.75/61.26	CNERG-BERT: 70
LC3: Hate expressed using profanity	140	1,064	6,394	alex-BERT: 5,869/98	alex-BERT: 78.69/70.00	alex-BERT: 51
				CNERG-BERT: 1,115/93	CNERG-BERT: 14.95/66.43	CNERG-BERT: 69
LC4: Non-Hateful use of profanity	100	1,478	7,709	alex-BERT: 1,683/1	alex-BERT: 18.32/1.00	alex-BERT: 49
				CNERG-BERT: 5,160/1	CNERG-BERT: 56.17/1.00	CNERG-BERT: 120
LC5: Hate expressed through reference in subsequent clauses	140	11,968	43,641	alex-BERT: 37,022/108	alex-BERT: 66.58/77.14	alex-BERT: 793
				CNERG-BERT: 30,276/93	CNERG-BERT: 54.44/66.43	CNERG-BERT: 855
LC6: Hate expressed through reference in subsequent sentences	133	11,968	42,416	alex-BERT: 35,958/101	alex-BERT: 66.12/75.94	alex-BERT: 783
				CNERG-BERT: 31,195/69	CNERG-BERT: 57.36/51.88	CNERG-BERT: 721
LC7: Hate expressed using negated positive statement	140	39,783	220,483	alex-BERT: 222,574/109	alex-BERT: 85.52/77.86	alex-BERT: 2,457
				CNERG-BERT: 152,929/116	CNERG-BERT: 58.76/82.86	CNERG-BERT: 4,365
LC8: Non-hate expressed using negated hateful statement	133	17,796	133,756	alex-BERT: 23,027/13	alex-BERT: 15.19/9.77	alex-BERT: 1,265
				CNERG-BERT: 113,265/26	CNERG-BERT: 74.74/19.55	CNERG-BERT: 1,626
LC9: Hate phrased as a question	140	11,864	101,569	alex-BERT: 98,879/107	alex-BERT: 87.17/76.43	alex-BERT: 961
				CNERG-BERT: 33,589/123	CNERG-BERT: 29.61/87.86	CNERG-BERT: 1,305
LC10: Hate phrased as a opinion	133	11,864	87,996	alex-BERT: 84,221/100	alex-BERT: 84.34/75.19	alex-BERT: 999
				CNERG-BERT: 27,637/109	CNERG-BERT: 27.68/81.95	CNERG-BERT: 1,348
LC11: Neutral statements using protected group identifiers	126	6	12	alex-BERT: 16/9	alex-BERT: 88.89/7.14	alex-BERT: 0
				CNERG-BERT: 1/0	CNERG-BERT: 5.56/0.00	CNERG-BERT: 0
LC12: Positive statements using protected group identifiers	189	57	246	alex-BERT: 151/23	alex-BERT: 49.83/12.17	alex-BERT: 7
				CNERG-BERT: 73/16	CNERG-BERT: 24.09/8.47	CNERG-BERT: 1
LC13: Denouncements of hate that quote it	173	23,728	167,404	alex-BERT: 20,511/17	alex-BERT: 10.73/9.83	alex-BERT: 1,229
				CNERG-BERT: 117,788/5	CNERG-BERT: 61.63/2.89	CNERG-BERT: 2,440
LC14: Denouncements of hate that make direct reference to it	141	17,796	127,067	alex-BERT: 17,060/4	alex-BERT: 11.78/2.84	alex-BERT: 1,070
				CNERG-BERT: 100,848/7	CNERG-BERT: 69.62/4.96	CNERG-BERT: 1,594

Self-BLEU and production rule coverage scores over all linguistic capabilities and 5 trials on the left and right, respectively. The results show that S^{2}LCT's test suite is more diverse than the baselines', with significantly lower production rule coverage scores and

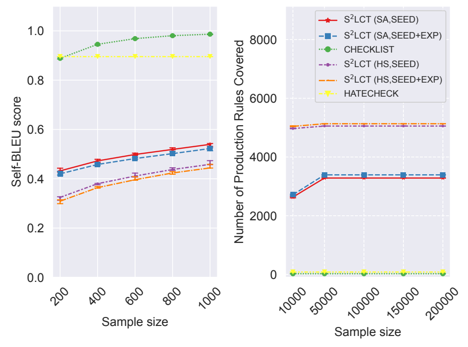


Figure 4: Results of Self-BLEU (left) and Production Rule Coverage (right) of S^{2}LCT and capability-based testing baselines for sentiment analysis (SA) and hate speech detection (HS). Use of only S^{2}LCT seed sentences and all S^{2}LCT sentences are denoted as SEED and SEED+EXP respectively.

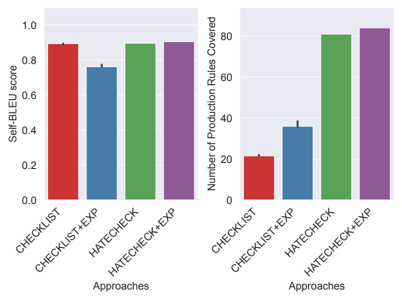


Figure 5: Results of Self-BLEU (left) and Production Rule Coverage (right) between original sentences of capability-based testing baselines and S^{2}LCT generated sentences from the original sentences.

Table 5: Comparison results against MT-NLP

Task	Approach	#Gen	Self-BLEU	PDRCov
SA	*S^{2}LCT*	**606**	**0.75± 0.01**	**338.8±12.03**
	MT-NLP	23	0.91± 0.0	96.0±0.0
HSD	*S^{2}LCT*	**800**	**0.69± 0.02**	**400.4±17.21**
	MT-NLP	211	0.79± 0.02	344.0±15.86

Table 6: Comparison results against adversarial attacks

Approach	#Gen	Self-BLEU	PDRCov
S^{2}LCT	**323**	0.435±0.005	**262.0±2.739**
Alzantot	20	**0.373±0.0**	170.0±0.0
BERT-Attack	25	0.438±0.0	178.0±0.0
PSO	25	0.411±0.0	178.0±0.0

significantly higher Self-BLEU scores. This highlights the advantages of searching from a real-world dataset rather than relying on limited preset templates. Furthermore, using expanded sentences in S^{2}LCT leads to lower Self-BLEU and higher production rule coverage scores, demonstrating the syntax-based expansion of S^{2}LCT improves sentence diversity.

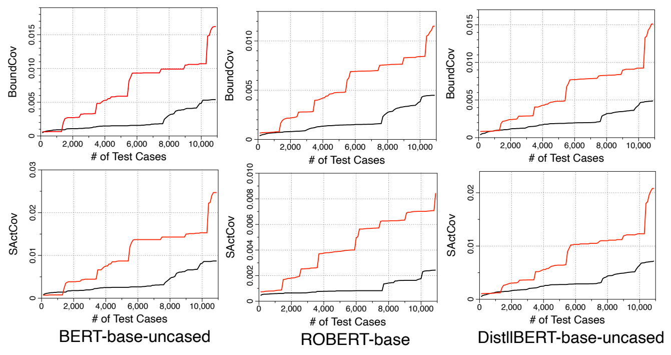


Figure 6: Coverage results of S^{2}LCT and CHECKLIST.

Figure 5 shows the Self-BLEU and production rule coverage scores of test suites generated by baseline methods and their expanded versions using S^{2}LCT. The x-axis shows the method name, and the y-axis shows the corresponding scores across all linguistic capabilities. The left sub-figure displays the Self-BLEU scores, and the right sub-figure shows the production rule coverage scores. The results indicate that the expanded CHECKLIST and Hatecheck achieve better production rule coverage scores than their original versions, demonstrating the effectiveness of S^{2}LCT's syntax-based expansion module in increasing the diversity of the generated test suite. Additionally, the expanded CHECKLIST performs worse in terms of Self-BLEU scores, while the expanded Hatecheck has comparable scores to its original version.

Table 5 compares S^{2}LCT's expanded sentences and MT-NLP for 100 randomly selected seeds. The first column lists the NLP task, and the second column displays the approaches for text generation. Columns 3-5 show the number of generated sentences, Self-BLEU, and production rule coverage scores over 5 sampling trials. S^{2}LCT generates more sentences than MT-NLP for all tasks and has higher Self-BLEU and production rule coverage scores, demonstrating the effectiveness of S^{2}LCT's syntax expansion in increasing test case diversity. MT-NLP sometimes fails to mutate seed sentences due to the unavailability of suitable human-related words for mutation.

Table 6 compares S^{2}LCT-expanded sentences with adversarial text generation baselines from Section 4.1. The first column shows the approach type and the second column shows the number of generated sentences using S^{2}LCT and the baselines from 50 randomly selected seeds. The third and fourth columns show the Self-BLEU and production rule coverage scores over 5 sampling trials. Results show that Alzantot et al. [1] has the lowest Self-BLEU scores, whereas S^{2}LCT expansion achieves the highest scores in the number of generated sentences and production rule coverage, introducing various syntax components while maintaining text diversity.

Neuron Coverage. Figure 6 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 6 represents the results for one sentiment analysis model. The first row is the *BoundCov* results and the second row is the *SActCov* results. We made three observations from the results. 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.

5.2 RQ2: Effectiveness

Our results show that *S^{2}LCT generates diverse test cases that expose classification errors in NLP models, outperforming the baselines.*
The Number of Test Cases. Table 3 and 4 present the results of our effectiveness metrics defined in Section 4.2. For all linguistic capabilities, S^{2}LCT generates a significant number of test cases, ranging from 70 on LC1 to 533,575 on LC8. For LC1, LC2, LC4, and LC5, S^{2}LCT generates fewer test cases than CHECKLIST because of the small number of seeds. However, the syntax-based sentence expansion phase generated 51 to 503 test cases. In Table 4, S^{2}LCT generates more test cases than Hatecheck for every linguistic capabilities except LC11, indicating that S^{2}LCT is more beneficial in generating a sufficient number of test cases. The results show that S^{2}LCT generates many test cases in the NLP models that fail to predict the correct labels, providing further qualitative test cases than baselines for finding errors.
Fail Rate and Failed Cases. Figure 3 demonstrates that at least one model introduces a higher number of failed test cases on S^{2}LCT test cases than CHECKLIST in 7 linguistic capabilities, and at least one model achieves a higher failure rate on S^{2}LCT than on CHECKLIST in all other linguistic capabilities (ranging from 4.27% to 99.64%) except for LC8 and LC9. Additionally, in table 4, we observe that every linguistic capabilities for hate speech detection has a higher number of failed test cases on S^{2}LCT test cases than Hatecheck, with the failure rate being higher for at least one model in every linguistic capabilities except for LC1 and LC5 (ranging from 1.89% to 88.89%). Based on these findings, we can conclude that S^{2}LCT is more effective in generating test cases to identify errors.
Pass-to-Fail Cases. We observed that many test cases failed in the expanded set but not in their corresponding seeds (as shown in the last column of Table 3 and 4). This type of error case ranges from 0 to 12,729 for sentiment analysis and from 0 to 4,365 for hate speech detection. These results demonstrate that the syntax-based sentence expansion phase effectively introduces more diverse sentence structures, which can potentially expose errors in NLP models that may not be evident in the original seed test cases.

Table 7: Consistency Results

Task	Type	#TC	LabelCons	LCRel	ExpValidity
SA	SEED	100	0.83	0.92	-
	EXP	100	0.83	0.92	1.00
HSD	SEED	100	0.80	0.84	-
	EXP	100	0.79	0.84	0.97

5.3 RQ3: Consistency

Table 7 shows the results of our consistency study. The first column lists the NLP tasks, and the second column distinguishes between seed and expanded sentences. The third column indicates the number of test cases used. Columns 4-6 present the scores of label consistency, LC relevancy, and expansion validity sentences, respectively. Our analysis shows that *S^{2}LCT generates test cases with high label consistency*, with scores of 0.83 for both seed and expanded test cases for sentiment analysis and 0.80 and 0.79 for seed and expanded cases, respectively, for hate speech detection, indicating that the test oracles constructed by S^{2}LCT align with human sentiment labeling most of the time. Moreover, the results show high expansion validity scores of 1.0 for sentiment analysis and 0.97 for hate speech detection, indicating that the tool effectively preserves the semantic meaning of seed sentences during the expansion process. The linguistic capability relevancy score is presented in column 5 of Table 7. 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 0.92 and 0.84 for sentiment analysis and hate speech detection, respectively, achieving high agreement with human assessment. The fact that the expanded sentences generated by S^{2}LCT have the same level of linguistic capability relevancy as the seed sentences demonstrates that the syntax-based sentence expansion retains the linguistic capabilities.

6 Application of S^{2}LCT

In this section, we showcase how S^{2}LCT can be used in conjunction with explainable ML techniques to assist developers in identifying the root causes of bugs in 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 [5, 17, 49] 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}

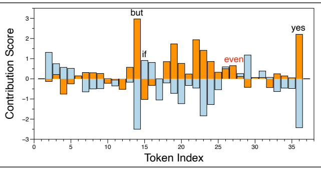
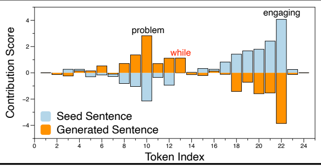
Seed Sentence	Generated Sentence	Score Visualization
Do I think that " Maybe it's asking too much, but if a great movie is truly going to inspire me, I want a little more than this . "? yes Do I think that " [MASK] [MASK] [MASK] too [MASK] , [MASK] if a [MASK] [MASK] is [MASK] [MASK] to [MASK] [MASK] , I want a [MASK] [MASK] [MASK] [MASK] [MASK] ? [MASK]	Do I think that " Maybe it's asking too much, but if a movie is truly going to inspire me, I want even a little more than this . "? yes [MASK] [MASK] [MASK] [MASK] Maybe it' [MASK] [MASK] but [MASK] [MASK] [MASK] a movie is truly going to inspire me [MASK] want even [MASK] [MASK] more than this . "? yes	
I wouldn't say, given The main problem being that it's only a peek., Quietly engaging. [MASK][MASK]'t[MASK][MASK][MASK][MASK][MASK][MASK][MASK][MASK][MASK]'s[MASK] a peek., Quietly engaging[MASK]	I wouldn't say, given The main problem being that while it's only a peek., Quietly engaging. I [MASK] t say, given The main problem being that while[MASK][MASK][MASK] a [MASK][MASK][MASK][MASK][MASK][MASK]	

Figure 7: Visualization of the buggy reason of two S^{2}LCT generated test cases.

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

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

To create such a template T, we first compute the contribution score of each input token using an existing explainable ML technique [17]. 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 7 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".

7 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 [26]. Second, using ranges of the sentiment scores in SST and majority voting method in HateXplain to determine the sentiment labels of search dataset may introduce incorrect labels. We performed a manual study to confirm that this is rare.

8 Related Work

In addition to the capability-based testing works discussed in Section 2, we review other related works in this section.

NLP Algorithms & Applications. Deep neural networks (DNNs) have significantly improved various natural language processing (NLP) applications [8, 10, 15, 27, 41, 45, 47, 52, 57, 61], including reading comprehension, hate speech detection, and machine translation. For instance, Word2vec [41] distributes the semantic of words into numeric vectors, which are then utilized to train neural networks for classification tasks. Meanwhile, Seq2Seq [57] presents an encoder-decoder neural network architecture that has been widely adopted for modeling the sequence generation task, particularly in machine translation applications. In recent years, Google [61] has introduced the attention mechanism, which, when combined with Seq2Seq, can greatly enhance the accuracy of the generated texts.

Machine Learning Testing & NLP Testing. Machine learning has shown great potential in various real-world applications. Nonetheless, despite the high accuracy rates of ML models, there have been instances where ML models can generate inferior results, leading to fatal accidents [31, 32]. Therefore, researchers have developed a series of techniques [6, 8, 11–14, 21, 22, 28, 36, 37, 44, 46, 48, 58, 62, 65, 69] to test ML-based applications. In recent years, researchers have investigated the occurrence of bugs produced by neural networks in NLP applications [7, 12, 13, 18, 20, 21, 33, 35, 38, 42, 48, 56, 64, 67, 69], inspired by the work on adversarial examples in computer vision. Our approach differs from existing work in that we concentrate on testing the linguistic capabilities of NLP applications in an automatic manner, a topic that has yet to be explored.

9 Conclusions

We introduce S^{2}LCT, which automatically generates test cases for testing NLP models. We evaluate the effectiveness of S^{2}LCT on two popular NLP tasks. Our study results reveal that the diversity of S^{2}LCT's test cases improves model coverage and reliability. Additionally, we analyze failure-inducing cases to identify bug causes, demonstrating the correctness and utility of S^{2}LCT for model evaluation.

References

[1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[2] Muhammad Hilmi Asyrofi, Zhou Yang, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, and David Lo. Biasfinder: Metamorphic test generation to uncover bias for sentiment analysis systems. *IEEE Transactions on Software Engineering*, 48(12):5087–5101, 2022.

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio 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, May 2010. European Language Resources Association (ELRA).

[4] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2014.

[5] Simin Chen, Soroush Bateni, Sampath Grandhi, Xiaodi Li, Cong Liu, and Wei Yang. *DENAS: Automated Rule Generation by Knowledge Extraction from Neural Networks*, page 813–825. Association for Computing Machinery, New York, NY, USA, 2020.

[6] Simin Chen, Mirazul Haque, Cong Liu, and Wei Yang. Deepperform: An efficient approach for performance testing of resource-constrained neural networks. In *37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[7] Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. Nmtsloth: understanding and testing efficiency degradation of neural machine translation systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1148–1160, 2022.

[8] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15365–15374, 2022.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 477–487, 2019.

[12] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. On adversarial examples for character-level neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 653–663, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[13] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[14] Alessio Gambi, Marc Mueller, and Gordon Fraser. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 318–328, 2019.

[15] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[16] Mor Geva, Yoav Goldberg, and Jonathan 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.

[17] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. Lemna: Explaining deep learning based security applications. In *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 364–379, 2018.

[18] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. Machine translation testing via pathological invariance. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 863–875, 2020.

[19] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. Annotation artifacts in natural language inference data. *arXiv preprint arXiv:1803.02324*, 2018.

[20] Pinjia He, Clara Meister, and Zhendong Su. Structure-invariant testing for machine translation. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 961–973. IEEE, 2020.

[21] Pinjia He, Clara Meister, and Zhendong Su. Testing machine translation via referential transparency. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 410–422. IEEE, 2021.

[22] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15262–15271, 2021.

[23] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[24] Jen-tse Huang, Jianping Zhang, Wenxuan Wang, Pinjia He, Yuxin Su, and Michael R Lyu. Aeon: A method for automatic evaluation of nlp test cases. *arXiv preprint arXiv:2205.06439*, 2022.

[25] HuggingFace. Huggingface, 2022.

[26] Mujtaba Husnain, Malik Muhammad Saad Missen, Nadeem Akhtar, Mickaël Coustaty, Shahzad Mumtaz, and VB Prasath. A systematic study on the role of sentiwordnet in opinion mining. *Frontiers of Computer Science*, 15(4):1–19, 2021.

[27] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[28] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1039–1049. IEEE, 2019.

[29] Nikita Kitaev, Steven Cao, and Dan Klein. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy, July 2019. Association for Computational Linguistics.

[30] Nikita Kitaev and Dan 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)*, pages 2676–2686, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[31] Fred Lambert. Understanding the fatal tesla accident on autopilot and the nhtsa probe. *Electrek, July*, 1, 2016.

[32] Sam Levin. Tesla fatal crash:'autopilot'mode sped up car before driver killed, report finds. *The Guardian*, 8, 2018.

[33] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*, 2018.

[34] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. BERT-ATTACK: adversarial attack against BERT using BERT. *CoRR*, abs/2004.09984, 2020.

[35] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. BERT-ATTACK: Adversarial attack against BERT using BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202, Online, November 2020. Association for Computational Linguistics.

[36] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang 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*, pages 120–131, 2018.

[37] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*, pages 100–111. IEEE, 2018.

[38] Pingchuan Ma, Shuai Wang, and Jin Liu. Metamorphic testing and certified mitigation of fairness violations in nlp models. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 458–465. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[39] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, jun 1993.

[40] Binny Mathew, Punyajoy Saha, Seid Muhie Yimam, Chris Biemann, Pawan Goyal, and Animesh Mukherjee. Hatexplain: A benchmark dataset for explainable hate speech detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14867–14875, 2021.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[42] John X. Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. Textattack: A framework for adversarial attacks in natural language processing. *CoRR*, abs/2005.05909, 2020.

[43] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[44] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, oct 2017.

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

[46] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. Cradle: cross-backend validation to detect and localize bugs in deep learning libraries. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1027–1038. IEEE, 2019.

[47] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[48] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy, July 2019. Association for Computational Linguistics.

[49] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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*, pages 1135–1144, 2016.

[50] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *Association for Computational Linguistics (ACL)*, 2020.

[51] Paul Röttger, Bertram Vidgen, Dong Nguyen, Zeerak Waseem, Helen Margetts, and Janet B Pierrehumbert. Hatecheck: Functional tests for hate speech detection models. *arXiv preprint arXiv:2012.15606*, 2020.

[52] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[53] Alan Searleman. A review of right hemisphere linguistic capabilities. *Psychological Bulletin*, 84(3):503, 1977.

[54] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, 2013. Association for Computational Linguistics.

[55] Ezekiel O. Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. Astraea: Grammar-based fairness testing. *CoRR*, abs/2010.02542, 2020.

[56] Zeyu Sun, Jie M Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. Automatic testing and improvement of machine translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 974–985, 2020.

[57] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[58] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[59] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Automated directed fairness testing. *CoRR*, abs/1807.00468, 2018.

[60] Sakshi Udeshi and Sudipta Chattopadhyay. Grammar based directed testing of machine learning systems. *CoRR*, abs/1902.10027, 2019.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[62] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 146–157, 2019.

[63] Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online, July 2020. Association for Computational Linguistics.

[64] Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online, July 2020. Association for Computational Linguistics.

[65] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2020.

[66] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 132–142, 2018.

[67] Xinze Zhang, Junzhe Zhang, Zhenhua Chen, and Kun He. Crafting adversarial examples for neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1967–1977, Online, August 2021. Association for Computational Linguistics.

[68] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. *SIGIR*, 2018.

[69] Wei Zou, Shujian Huang, Jun Xie, Xinyu Dai, and Jiajun Chen. A reinforced generation of adversarial examples for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3486–3497, 2020.