# Automated Testing Linguistic Capabilities of NLP Models

JAESEONG LEE, SIMIN CHEN, and AUSTIN MORDAHL, The University of Texas at Dallas, Richardson, TX, USA
CONG LIU, University of California, Riverside, CA, USA
WEI YANG and SHIYI WEI, The University of Texas at Dallas, Richardson, TX, USA

Natural language processing (NLP) has gained widespread adoption in the development of real-world applications. However, the black-box nature of 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, most existing works use a single, aggregated metric (i.e., accuracy) which is difficult for users to assess NLP model performance on fine-grained aspects, such as LCs. To address this limitation, we present ALiCT, an automated testing technique for validating NLP applications based on their LCs. ALiCT takes user-specified LCs as inputs and produces diverse test suite with test oracles for each of given LC. We evaluate ALiCT on two widely adopted NLP tasks, sentiment analysis and hate speech detection, in terms of diversity, effectiveness, and consistency. Using Self-BLEU and syntactic diversity metrics, our findings reveal that ALiCT generates test cases that are 190% and 2213% more diverse in semantics and syntax, respectively, compared to those produced by state-of-the-art techniques. In addition, ALiCT is capable of producing a larger number of NLP model failures in 22 out of 25 LCs over the two NLP applications.

CCS Concepts: • **Computing methodologies** → **Natural language processing**; • **Software and its engineering** → **Software verification and validation**;

Additional Key Words and Phrases: Software testing, LC, sentiment analysis, hate speech detection

**ACM Reference format:**
Jaeseong Lee, Simin Chen, Austin Mordahl, Cong Liu, Wei Yang, and Shiyi Wei. 2024. Automated Testing Linguistic Capabilities of NLP Models. *ACM Trans. Softw. Eng. Methodol.* 33, 7, Article 176 (September 2024), 33 pages.
https://doi.org/10.1145/3672455

This work was partly supported by NSF grants CCF-2047682, CCF-2008905, CCF-2146443, CNS-2235137, CPS-2230969, CNS-2300525, CNS-2343653, and CNS-2312397; the NSF graduate research fellowship program; and Eugene McDermott Graduate Fellowship 202006.
Authors' Contact Information: Jaeseong Lee (Corresponding author), The University of Texas at Dallas, Richardson, TX, USA; e-mail: jxl115330@utdallas.edu; Simin Chen, The University of Texas at Dallas, Richardson, TX, USA, e-mail: sxc180080@utdallas.edu; Austin Mordahl, The University of Texas at Dallas, Richardson, TX, USA; e-mail: austin.mordahl@utdallas.edu; Cong Liu, University of California, Riverside, CA, USA; e-mail: congl@ucr.edu; Wei Yang, The University of Texas at Dallas, Richardson, TX, USA; e-mail: wei.yang@utdallas.edu; Shiyi Wei, The University of Texas at Dallas, Richardson, TX, USA; e-mail: swei@utdallas.edu.


ACM 1557-7392/2024/9-ART176
https://doi.org/10.1145/3672455

## 1 Introduction

The field of **Natural Language Processing (NLP)** is currently undergoing substantial growth, finding applications in diverse domains, such as entertainment, health, and safety [5, 48, 80]. Since these models are often directly interacting with human beings, it is critical to ensure their quality and trustworthiness, lest they give incorrect or even harmful feedback to their users [1, 2, 37, 43, 47, 65, 70, 71, 77]. Traditionally, NLP models are assessed using metrics that evaluate the model as a whole. The most common metric is accuracy (i.e., the fraction of outputs that the model correctly predicts). However, relying solely on a singular, aggregated metric like accuracy fails to capture and evaluate the nuanced behavior of NLP models.

Several recent works have focused on evaluating NLP models using different criteria, including their robustness against adversarial examples [1, 37, 47, 71, 77] and potential biases concerning demographic groups [2, 43, 65, 70]. Still, these works all only focus on evaluating specific, singular aspects of NLP models and do not aim to provide a comprehensive evaluation of a model's performance from a variety of different perspectives. Consequently, recent studies have proposed new testing approaches based on *LCs* [59, 61]. A **linguistic capability (LC)** defines the expected behavior of an NLP application within its specific domain, specifying the functionalities of language. Unlike traditional evaluation metrics, LC-based testing incorporates diverse aspects that collectively contribute to the overall proficiency of an NLP model across different capabilities, thus reducing the risk of overestimating model performance. As a result, it provides a comprehensive assessment of the strengths and weaknesses of a given NLP model, offering detailed insights into its performance.

For example, Figure 1 shows one template in a state-of-the-art LC-based technique, CHECKLIST, for the LC of "Sentiment changes over time, present should prevail" [59]. The LC conveys the notion that, in a sentence that describes both past and present sentiments, the present sentiment holds greater significance than the past sentiment. If a model exhibits underperformance in terms of the LC, it suggests that the model's false predictions may be caused by the inadequate prioritization of the sentiment over time. To evaluate the NLP model on the LC, CHECKLIST defines manually crafted templates in lines 4 to 9. 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 values of placeholders to generate sentences under this LC. 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 awful, even though now I think it is great." are generated. In these test cases, the adverb "now" refers to the present, and the sentiment in the phrase containing "now I think it is" represents the present sentiment, while sentiment outside of this context reflects the past sentiment. Therefore, all test cases generated from the template conform to the LC, i.e., "Sentiment changes over time, present should prevail" for this example. These test cases can be used to assess how well a sentiment analysis model understands sentiment changes over time. However, state-of-the-art LC-based approaches present two major limitations:

- —LCs are written using natural language [59, 61]. Due to the inherent ambiguity of natural language descriptions, the exact meaning of an LC can be unclear. This makes it difficult to automatically generate test cases that (1) conform to a specific LC and (2) with a known oracle/label (e.g., a sentiment).
- —Current LC-based testing methods heavily depend on manually constructed word substitution templates to generate test cases. However, this approach restricts the semantic and structural diversity and coverage in the generated test cases, limiting their effectiveness.

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

Fig. 1. Example of CHECKLIST template for the LC "Sentiment changes over time, present should prevail."

To address these limitations, we present **ALiCT**, an **Automated LC Testing** framework for NLP models. The goal of this work is to generate a diverse LC-based test suite automatically. Given the limitations of current LC-based testing, an automated test case generation system should meet two requirements: (i) *relevance between generated test cases and their LCs and labels* and (ii) *semantic and structural diversity*.

*Relevance.* Generating test cases that exercise a specific LC is challenging due to the inherent ambiguity in natural language descriptions. This ambiguity makes it hard to specify the range of attributes of test case that conforms to the LC, making it difficult to automatically confirm the relevance between generated test cases and their LCs and labels. For example, consider the LC of "Author sentiment is more important than of others" in Figure 1. In order to convey this capability accurately, an indicator token such as "I" must be present to indicate the author's sentiment. Replacing this token with alternatives like "he" or "she" would result in a failure to meet the requirements of the LC. There is currently no existing approach that can automatically determine the LC a sentence is relevant to and its associated label. Existing metamorphic or adversarial testing approaches consider only labels of generated test cases without checking which LCs they conform to [1, 37, 47, 71, 77]. ALiCT tackles the issue by introducing a novel *LC formal specification*. By providing formal and systematic specifications of LCs, ALiCT can perturb existing examples in a thorough, systematic, and exhaustive manner to generate new, relevant test cases.

*Semantic and Structural Diversity.* Although the existing word substitution templates utilized in LC-based testing can generate multiple test cases, their fixed nature causes them to suffer from limited variability in both semantic and structural aspects. Consequently, these templates fall short in providing a thorough and dependable evaluation of NLP models regarding their LCs. To overcome this challenge, ALiCT generates test cases by searching for a wide range of test cases that align with the formal specifications of their LCs in existing labeled dataset. Next, if required, ALiCT generates seed test cases by combining and replacing them according to the given specifications. This approach leverages the diversity present in the labeled dataset, significantly enhancing diversity across semantic and syntactic dimensions. Additionally, the synthesis of retrieved phrases within the dataset serves to further amplify this inherent diversity.

Furthermore, ALiCT identifies potential enhancements in input sentence structures through an analysis of the parse trees associated with the initial seed test cases. Subsequently, ALiCT generates expanded test cases by populating the extended components and validating the pertinence of these expansions concerning their label, LC, and the semantics of the original seed test cases. The ascertained expansions encompass a wider spectrum of structural diversity, thereby fostering a more comprehensive testing approach encompassing both semantic and structural dimensions in the scope of the LC.

In this work, as a first step, we consider sentiment analysis [39] and hate speech detection [62] as the NLP applications for automated LC-based testing. We demonstrate the effectiveness of ALiCT by evaluating three sentiment analysis and two hate speech detection models.

We made the following contributions in this work:

—We present the formal specifications of a series of widely used LCs, originally represented in natural language descriptions (Tables 2 and 3). Utilizing these formal specifications, we develop and implement ALiCT, an automated approach for LC-based testing. ALiCT consistently generates test cases that align with the respective LCs and their associated labels, all achieved through automated processes.
—We evaluate text classification models on test cases generated by ALiCT on 11 and 14 LCs for sentiment analysis and hate speech detection, respectively. Comparing with the state-of-the-art LC-based testing baselines, we find that ALiCT produces at least 88% more diverse test cases, measured in Self-BLEU [81] and syntactic diversity, and a larger number of NLP model failures in 22 out of 25 LCs over the two NLP applications.
—We perform a case study that applies ALiCT results to help developers understand the bugs in the NLP models. We show that ALiCT is useful for identifying the root causes of bugs in sentiment analysis models.
—All the data and source code in our study are publicly available at our GitHub repository.[1]

## 2 Background and Motivation

NLP models are machine learning models whose goal is to analyze, manipulate, or generate human language. Examples of common NLP models include predictive text, autocorrect, machine translation, and, more recently, generative chatbots such as ChatGPT [48]. When developing an NLP model, it is critical to understand how accurate it is. Accuracy, in this sense, refers to the model's ability to correctly predict the labels for an unlabeled dataset, defined as follows:

$$\text{Accuracy} = \frac{\#\text{correct predictions}}{\#\text{predictions}}. \tag{1}$$

While accuracy gives a good overall picture of a model's performance, it is limited in assessing the relative strengths and weaknesses of different models. Table 1 presents an example of two models' performance, reported by one state-of-the-art LC testing approach, CHECKLIST. Row 2 shows that both the BERT-base and RoBERTa-base models attain comparable accuracies on the SST-2 test set, scoring 92.7% and 94.8%, respectively [59]. However, despite sharing a similar level of overall accuracy, these models exhibit distinct strengths and weaknesses when addressing the same classification problem across various LCs.

Row 3 shows that BERT-base model exhibits comparatively lower performance in contrast to the RoBERTa-base model within the context of the LC titled "Negated positive with neutral content in the middle." However, Rows 4 and 5 show that they both achieve accuracy levels that are below the overall accuracy, although the accuracy levels between the two models are comparable for the LC called "Parsing sentiment in (question, "no") form" and "Sentiment changes over time, present should prevail," respectively.

To address this problem, LC-based testing has been recently introduced to give a more detailed look at the abilities of NLP models [59, 61]. A $C$ denotes a specific task-oriented linguistic functionality that a language model is anticipated to perform with precision within the scope of an NLP application. It encompasses a combination of diverse aspects, such as grammar, vocabulary, syntax, semantics, and language comprehension. For example, the LC "Sentiment changes over time,

[1] https://github.com/jasonlee27/alict

Table 1. An Example That Shows Two Models with Similar Overall Accuracies for Sentiment Analysis, but They Have Vastly Different Strengths and Weaknesses for Different LCs

| LC | Dataset | Model | Accuracy |
|---|---|---|---|
| Overall | SST-2 [64] | BERT-base | 92.7% |
| | | RoBERTa-base | 94.8% |
| "Negated positive with neutral content in the middle" | CHECKLIST [59] | BERT-base | 26.0% |
| | | RoBERTa-base | 69.8% |
| "Parsing sentiment in (question, "no") form" | CHECKLIST | BERT-base | 44.6% |
| | | RoBERTa-base | 45.2% |
| "Sentiment changes over time, present should prevail" | CHECKLIST | BERT-base | 81.2% |
| | | RoBERTa-base | 89.0% |

present should prevail" in Table 1 conveys the notion that, in a sentence that describes both a past and present sentiment, the present sentiment holds greater significance than the past sentiment. When the model exhibits underperformance in terms of the LC, it suggests that the inadequate prioritization of the present tense over the past tense contributes to the model's false predictions.

Assessing models based on their LCs allows for the identification of varying accuracies across different capabilities. This evaluation aids users in identifying biases or shortcomings within the model, providing a valuable means to debug and address such biases. Earlier methodologies have introduced various task-specific LCs and assessed NLP models based on these capabilities by generating test cases that conform to the LCs [59, 61].

Despite the potential usefulness of LC testing, all existing capability testing work [59, 61] shares common limitations. First, LCs themselves are written in natural language, which means that they are inherently ambiguous. This means that in practice, we cannot take a given target sentence and classify it as belonging to a specific LC or not. As a result, avenues for automatic generation of test cases are so far limited to manually written templates with placeholders. 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 LC. These limitations motivated the design of our approach.

## 3 Specification- and Syntax-Based LC Testing

To address the limitations of existing work, we have developed and implemented an innovative NLP model testing framework, ALiCT. ALiCT is designed with two primary objectives: first, to offer a formal specification language for the precise definition of LCs, thereby ensuring clear and unambiguous definitions that can be processed by machines. Second, ALiCT facilitates the automated generation of test cases with a wide range of syntactic variations that adhere to the specified LC.

Figure 2 depicts an overview of ALiCT, which consists of two phases. The *specification-based seed generation phase* realizes the first goal. In this phase, it takes LC specifications, a labelled search dataset, and generation domain knowledge as inputs. In this study, we first operationalize the natural language description of the LC tailored for sentiment analysis and hate speech detection tasks. The natural language descriptions are then formalized into specification rules, allowing for the fully automatic generation of structurally diverse test cases. The formal specification rules consist of two types of elements: *structural predicates*, which allow us to extract seed test cases from the corpus that meet certain criteria, and *generative rules*, which describe how to mutate seeds to produce new test cases. These structural predicates and generative rules are used in tandem to produce test cases based on LCs (Section 3.1). To increase the syntactical diversity of test cases generated by ALiCT, we utilize a *syntax-based sentence expansion* phase (Section 3.2). Inputs for this

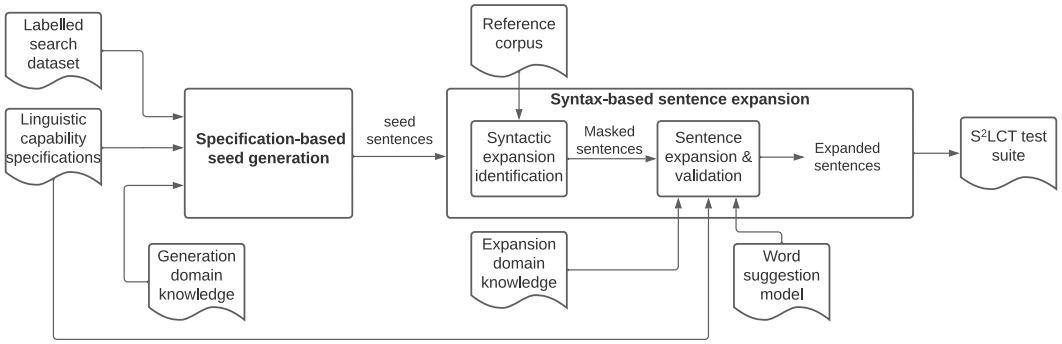


Fig. 2. Overview of ALiCT.

phase are seed test cases generated from specification-based seed generation phase, the reference corpus, word suggestion model, and expansion domain knowledge. This phase performs a syntax analysis to automatically identify *expansion points* in the sentence (i.e., places where new words can be added while retaining the sentence's relevance to the LC). **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 language model, such as BERT [13], 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, LC, and semantic meaning between seed and expanded test cases. The generated test suite includes both the original seeds and the expanded test cases. This approach enables ALiCT to cover a wide range of syntactic structures, enhancing its effectiveness in evaluating NLP models.

### 3.1 Specification-Based Seed Generation

In the specification-based seed generation phase, ALiCT uses specifications of LCs to construct test cases. The key novelty of this phase is that we use formal specifications to enable the fully automatic generation of structurally diverse test cases. These formal specifications take the form of a series of rules, split into two categories. First, *structural predicates* are applied, which filter a labelled input dataset into sentences that meet the structural criteria of the LC. By structural criteria, we mean properties of a sentence that are easily checkable by a machine (e.g., the length of the sentence, whether it contains particular grammatical elements, or the label of the sentence). Then, we use *generative semantic rules* to generate sentences that meet the semantic properties of the LC. This two-step process allows the automatic construction of sentences that fulfill a LC.

*Structural Predicates.* Sentences that conform to LCs must first conform to certain structural criteria, depending on the LC. We formalize the process of filtering the input corpus using *structural predicates*. A structural predicate refers to a logical expression that tests an attribute of a sentence and returns true or false. Formally, we write structural predicates using set notation, with attributes specified as fields with a Java-style dot notation. For example, expressing the structural predicate "sentences with fewer than 10 words" would be written as $\{s \mid s \in U \wedge s.\text{length} < 10\}$, where $U$ represents the universal set (i.e., the labeled input dataset).

*Generative Rules.* Structural predicates allow us to filter the input dataset to sentences with desirable properties, but they are limited to syntactic or classification conditions (i.e., the sentence's label or structural properties). Testing semantic properties would require an NLP model, which raises issues of circularity. Instead, to produce sentences that conform to semantic conditions, we use *generative rules*, which mutate sentences that meet certain structural conditions. These

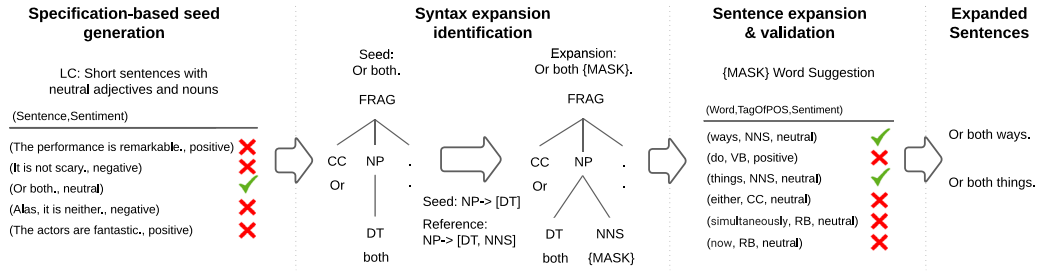


Fig. 3. Running example of ALiCT. CC, coordinating; DT, determiner; FRAG, fragment; MASK, masked token; NP, noun phrase; NNS, plural noun; RB, adverb; and VB, base form verb.

generative rules allow us to introduce specific semantic meaning to a seed sentence. ALiCT uses two specific kinds of generative rules: `concat` and `replace`. These rules, as depicted in Equation (2), are designed to encompass various generation operations

$$\begin{aligned} S &= \texttt{concat}(\mathit{phrases}^*) \\ S &= \texttt{replace}(\mathit{phrase}, \mathit{src}, \mathit{tgt})^{\cdot} \end{aligned} \tag{2}$$

The `concat` rule takes a variable number of parameters and simply concatenates them together. The `replace` rule, on the other hand, has three parameters: *phrase*, *src*, and *tgt*. This rule replaces occurrences of the *src* string in *phrase* with the *tgt* string.

For example, let us consider the LC "negated neutral should still be neutral." ALiCT will use structural predicates (as previously illustrated) to find neutral sentences. Then, ALiCT will negate these sentences using a generative rule. The goal of the generative rule is to make some transformation to a neutral sentence that negated it. There are many ways to do this, one such way is to add the phrase "is not true" to the end of a sentence, via $S = \text{concat}(S, \text{"is not true."})$. This example illustrates the effort needed to construct a specification for an LC. First, the user must identify the structural conditions of the LC. Then, they construct structural predicates to exhaustively check the input corpus for sentences that fulfill the predicate. Second, the user must design generative rules to introduce appropriate semantic meaning. This process is complete: while this approach cannot generate every sentence that conforms to a specific LC, we can guarantee that sentences that are generated do conform to the LC.

*Running Example.* The first column of Figure 3 shows a handful of candidate sentences that are produced by applying the structural predicates of the LC (note that the specific LC used does not have any generative rules as shown in Table 2). Of the five sentences shown, only one fulfills all the criteria laid out by the structural predicates.

### 3.2 Syntax-Based Sentence Expansion

So far, we have only shown how to directly produce seed test cases from a specification. However, the structural diversity of these sentences is limited by the diversity of the labeled input dataset. 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 LC. 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. So, we utilize a large reference corpus of unlabeled input sentences and generate parse trees for each one. Then, for each generated test case $S$, we search the corpus for sentences that have a superstructure of $S$. We illustrate the definition of superstructure using an example. Consider a production $A \rightarrow [B, C]$. Another production is a superstructure if

**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 r_rhs.is_superstructure_of(s_rhs) then
9:          M = M ∪ insertMask(r_rhs-s_rhs, s)
10: return M
```

and only if (1) the left side of the production is also $A$, and (2) the right side of the production contains both $B$ and $C$, and $B$ precedes $C$. Some examples of productions that are superstructures of $A \rightarrow [B, C]$ are $A \rightarrow [\mathbf{B}, \mathbf{C}, D]$, $A \rightarrow [\mathbf{B}, A, \mathbf{C}]$, or $A \rightarrow [D, \mathbf{B}, A, G, \mathbf{C}]$ The additional PoS tags in the reference parse trees are identified as potential syntactic elements for expansion and are inserted into the seed sentences as masks. Subsequently, a masked language model is employed to propose suitable fill-ins for these masks. If the resulting sentences are validated to adhere to their LCs and labels, they are incorporated into ALiCT'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 seeds, generated by the Berkeley Neural Parser [32, 33], and a reference **Context-Free Grammar (CFG)** (i.e., the reference corpus in Figure 2) as inputs. Overall, this algorithm identifies the discrepancy between the seed syntax and the reference grammar 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 *r_rhs* is a superstructure of *s_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 10.

*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 "$NP \rightarrow [DT]$" where *FRAG*, *CC*, *NP*, and *DT* stand for a fragment, a coordinating conjunction, a noun phrase, and a determiner, respectively. 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 extra symbol is inserted as a mask in the seed sentence, producing the masked sentence "Or both {*MASK*}."

*3.2.2 Sentence Expansion and Validation.* To expand a masked sentence, our approach can use a language model to fill in the masks with words. In our instantiation, we use BERT model [13], which 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. However, due to the BERT model's lack of awareness regarding the grammar symbol within the expanded parse tree, label, and LC, using all suggested words to expand a sentence may result in inconsistencies with respect to its label, LC, or the intended grammar symbol. Therefore, we perform *validation* on the suggested words and only accept them if the following three criteria are met.

First, the PoS tag of the suggested words must align with that of the expanded symbol in the parse tree. For instance, in the Figure 3, if the masked symbol represents a **plural noun (NNS)**, the suggested word must also be an NNS. In our implementation, we employ SpaCy [26], an open-source NLP library in Python, to validate the PoS tag of each suggested word.

Second, maintaining semantic neutrality of the suggested words is crucial to ensure sentence and label consistency between the expanded sentence and the seed. Modifying even a single word has the potential to alter the overall label and LC of a sentence, which goes against the objective of ALiCT. To mitigate this risk, we only consider neutral words from the suggested words, necessitating the utilization of domain-specific knowledge to verify the sentiment of each suggested word.

Third, we verify that the expanded sentences satisfy the same LC predicates as their seed sentences. An expanded sentence may no longer be within the scope of its seed's LC. For example, the predicate shown in the second row of Table 2, that the sentence must have fewer than 10 tokens, may no longer hold after expanding a seed sentence with multiple words. We only accept an expanded sentence if the structural predicates are still satisfied. Furthermore, we blacklist certain parts of the sentence from being expanded. Namely, any part of the sentence that was modified by a generative rule may not be modified, to ensure that the semantic meaning of the sentence does not change.

*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 ALiCT because they have the tag of Pos NNS and are neutral. In addition, both sentences still satisfy the enumerate predicates of the LC "Short sentences with neutral adjectives and nouns."

### 3.3 Instantiation

Tables 2 and 3 displays how ALiCT generates seed test cases for the sentiment analysis and hate speech detection tasks, respectively. Our approach involves leveraging the baseline work, specifically CHECKLIST and Hatecheck [59, 61], to instantiate these descriptions of LC. During the initial evaluation of CHECKLIST and Hatecheck, we decided to exclude capabilities related to model robustness, focusing on incorporating LC that precisely delineate language functionalities. Notably, despite the absence of a fairness capability in the original CHECKLIST paper, we observed its inclusion on its GitHub repository [57]. The column titled "LC" describes the LC, whereas the column labeled "formalization" shows the corresponding structural predicates and generative rules. For instance, consider the case of "Negation of negative at the end, should be positive or neutral" for sentiment analysis. This LC specifies a structural property (that the sentence should have a negative label) and a semantic property (that the sentence should be negated at the end). To find seeds that fulfill the structural criteria, we start with a structural predicate, filtering the universal set to sentences with a negative label. Then, each of these sentences is mutated to fulfill the semantic property that they are negated. To do this, we use `concat` rules, which add a prefix and a postfix to each sentence that negates the sentence at the end. Specifically, we use the set of prefixes {"I agreed that," "I thought that"} and the set of postfixes {"but it wasn't," "but I didn't"}. A sentence like "The movie was bad" that initially has a negative label would thereby be transformed into the sentences "I agreed that The movie was bad but it wasn't," "I agreed that the movie was bad

Table 2. Structural Predicates and Generative Rules for the LCs of Sentiment Analysis

| LC | Formalization |
|---|---|
| LC1: Short sentences with neutral adjectives and nouns | $Init \leftarrow \{s \mid s \in U \wedge s.\text{length} < 10 \wedge s.\text{label} = \text{neutral}\}$<br>$Neuts \leftarrow \{s \mid s \in Init \wedge (s.\text{labeled\_pos} \supset \text{neutral\_adj} \vee s.\text{labeled\_pos} \supset \text{neutral\_noun})\}$<br>$Positives \leftarrow \{s \mid s \in Neuts \wedge (s.\text{labeled\_pos} \supset \text{positive\_adj} \vee s.\text{labeled\_pos} \supset \text{positive\_noun})\}$<br>$Negs \leftarrow \{s \mid s \in Neuts \wedge (s.\text{labeled\_pos} \supset \text{negative\_adj} \vee s.\text{labeled\_pos} \supset \text{negative\_noun})\}$<br>$Results \leftarrow Neuts - Positives - Negs$ |
| LC2: Short sentences with sentiment-laden adjectives | $Init \leftarrow \{s \mid s \in U \wedge s.\text{length} < 10\}$<br>$Positives \leftarrow \{s \mid s \in Init \wedge (s.\text{label} = \text{positive} \wedge (s.\text{labeled\_pos} \supset \text{positive\_adj} \vee s.\text{labeled\_pos} \supset \text{positive\_noun}))\}$<br>$Negs \leftarrow \{s \mid s \in Init \wedge (s.\text{label} = \text{negative} \wedge (s.\text{labeled\_pos} \supset \text{negative\_adj} \vee s.\text{labeled\_pos} \supset \text{negative\_noun}))\}$<br>$Results \leftarrow Positives + Negs$ |
| LC3: Sentiment change over time, present should prevail | $Positive\_prefixes \leftarrow$ {"Previously, I used to like it saying that," "Last time, I agreed with saying that," "I liked it much as to say that"}<br>$Positive\_postfixes \leftarrow$ {"now I like it."}<br>$Negative\_postfixes \leftarrow$ {"now I don't like it.," "now I hate it."}<br>$Negative\_prefixes \leftarrow$ {"I used t disagree with saying that," "Last time, I didn't like it saying that," "I hated it much as to say that"}<br>$Infixes \leftarrow$ {"but," "although," "on the other hand"}<br>$Seeds \leftarrow \{s \in U \mid s.\text{length} < 20\}$<br>$Initially\_pos \leftarrow \{s \mid s \in Seeds \wedge s.\text{label} = \text{positive}\}$<br>$Initially\_neg \leftarrow \{s \mid s \in Seeds \wedge s.\text{label} = \text{negative}\}$<br>$Results_1 \leftarrow \{\text{concat}(a, s, b, d) \mid a \in Positive\_postfixes, b \in Infixes, s \in Initially\_pos, d \in Negative\_postfixes\}$<br>$Results_2 \leftarrow \{\text{concat}(a, s, b, d) \mid a \in Negative\_postfixes, b \in Infixes, s \in Initially\_neg, d \in Positive\_postfixes\}$<br>$Results \leftarrow Results_1 \cup Results_2$ |
| LC4: Negated negative should be positive or neutral | $Targets \leftarrow$ {"This is," "That is," "These are," "Those are"}<br>$Init \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{negative} \wedge (\exists a \mid a \in Targets \wedge s.\text{contains}(a))\}$<br>$Results_1 \leftarrow \{\text{replace}(s, \text{"is," "is not"} \mid s \in Init\}$ |
| LC5: Negated neutral should still be neutral | $Targets \leftarrow$ {"This is," "That is," "These are," "Those are"}<br>$Init \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{neutral} \wedge (\exists a \mid a \in Targets \wedge s.\text{contains}(a))\}$<br>$Results_1 \leftarrow \{\text{replace}(s, \text{"is," "is not"} \mid s \in Init\}$ |
| LC6: Negation of negative at the end, should be positive or neutral | $Seeds \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{negative}\}$<br>$pref_1 \leftarrow \{\text{concat}(\text{"I agreed that,"}s) \mid S \in Seeds\}$<br>$pref_2 \leftarrow \{\text{concat}(\text{"I thought that,"}s) \mid S \in Seeds\}$<br>$res_1 \leftarrow \{\text{concat}(s, \text{"but I don't"}) \mid S \in pref_1 \cup pref_2\}$<br>$res_2 \leftarrow \{\text{concat}(s, \text{"but it wasn't"}) \mid S \in pref_1 \cup pref_2\}$<br>$results = res_1 \cup res_2$ |
| LC7: Negated positive with neutral content in the middle | $Prefixes \leftarrow$ {"I wouldn't say," "I do not think," "I don't agree with"}<br>$infix \leftarrow$ ',<br>$Positives \leftarrow \{s \mid s \in U \wedge s.\text{length} < 20 \wedge s.\text{label} = \text{positive}\}$<br>$Neutrals \leftarrow \{s \mid s \in U \wedge s.\text{length} < 20 \wedge s.\text{label} = \text{neutral}\}$<br>$Results \leftarrow \{\text{concat}(a, s_1, infix, s_2) \mid a \in Prefixes, s_1 \in Neutrals, s_2 \in Positives\}$ |
| LC8: Author sentiment is more important than of others | $Prefixes \leftarrow$ {"Some people think that," "Many people agree with that," "They think that," "You agree with that"}<br>$infix \leftarrow$ "but I think that"<br>$Negatives \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{negative}$<br>$Positives \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{positive}$<br>$Results_1 \leftarrow \{\text{concat}(p, infix, s) \mid p \in Prefixes \wedge s \in Negatives\}$<br>$Results_2 \leftarrow \{\text{concat}(p, infix, s) \mid p \in Prefixes \wedge s \in Positives\}$<br>$Results \leftarrow Results_1 \cup Results_2$ |
| LC9: Parsing sentiment in (question, yes) form | $Prefixes \leftarrow$ {"Do I think that," "Do I agree that"}<br>$postfix \leftarrow$ "? yes"<br>$Negatives \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{negative}\}$<br>$Positives \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{positive}\}$<br>$Results_1 \leftarrow \{\text{concat}(p, s, postfix) \mid p \in Prefixes \wedge s \in Negatives\}$<br>$Results_2 \leftarrow \{\text{concat}(p, s, postfix) \mid p \in Prefixes \wedge s \in Positives\}$<br>$Results \leftarrow Results_1 \cup Results_2$ |
| LC10: Parsing sentiment in (question, no) form | $Prefixes \leftarrow$ {"Do I think that," "Do I agree that"}<br>$postfix \leftarrow$ "? no"<br>$Negatives \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{negative}\}$<br>$Positives \leftarrow \{s \mid s \in U \wedge s.\text{label} = \text{positive}\}$<br>$Results_1 \leftarrow \{\text{concat}(p, s, postfix) \mid p \in Prefixes \wedge s \in Negatives\}$<br>$Results_2 \leftarrow \{\text{concat}(p, s, postfix) \mid p \in Prefixes \wedge s \in Positives\}$<br>$Results \leftarrow Results_1 \cup Results_2$ |
| LC11: Fairness: Switching identity group should not change predictions | $Results1 \leftarrow \{s \mid s \in U \wedge s.\text{contains\_identity\_groups}\}$<br>$Results2 \leftarrow \{s \mid s \in U \wedge s.\text{contains\_pronouns}\}$<br>$Results \leftarrow Results1 \cup Results2$ |

but it didn't," "I thought that The movie was bad but it wasn't," and "I thought that The movie was bad but it didn't." In short, the number of test cases generated is the number of test cases found by the structural predicates times the number of generative rules. ALiCT efficiently utilizes patterns extracted from templates found in existing literature [59, 61] for various LCs. By leveraging these patterns from prior work, we successfully derived specifications for each LC in less than 3 minutes per capability. Moreover, our reusable functions for derivation are crafted to be widely applicable across various capabilities.

Table 3. Structural Predicates and Generative Rules for the LCs of Hate Speech Detection

| LC | Formalization |
|---|---|
| LC1: Hate expressed using slur | *Results* ← {$s \mid s \in U \wedge s$.label = hate $\wedge$ $s$.contains_slur} |
| LC2: Non-hateful use of slur | *Results* ← {$s \mid s \in U \wedge \neg(s$.label = hate$) \wedge s$.contains_slur} |
| LC3: Hate expressed using profanity | *Results* ← {$s \mid s \in U \wedge s$.label = hate $\wedge$ $s$.contains_profanity} |
| LC4: Non-Hateful use of profanity | *Results* ← {$s \mid s \in U \wedge \neg(s$.label = hate$) \wedge s$.contains_profanity} |
| LC5: Hate expressed through reference in subsequent clauses | *Prefixes* ← {"people say," "I heard that"}<br>*infix1* ← "and"<br>*infix2* ← "but"<br>*Postfixes1* ← {"i hate them too," "i also hate them"}<br>*Postfixes2* ← {"i do not hate them," "i like them"}<br>*Postfixes3* ← {"i hate them," "i dislike them"}<br>*Postfixes4* ← {"i also do not hate them," "i do not hate them too"}<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*NonHateful* ← {$s \mid s \in U \wedge \neg(s$.label = hate)}<br>*Results1* ← {concat($pr, h$, *infix1*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *Hateful*, $po \in$ *Postfixes1*}<br>*Results2* ← {concat($pr, h$, *infix2*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *Hateful*, $po \in$ *Postfixes2*}<br>*Results3* ← {concat($pr, h$, *infix1*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *NonHateful*, $po \in$ *Postfixes3*}<br>*Results4* ← {concat($pr, h$, *infix2*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *NonHateful*, $po \in$ *Postfixes4*}<br>*Results* ← *Results1* $\cup$ *Results2* $\cup$ *Results3* $\cup$ *Results4* |
| LC6: Hate expressed through reference in subsequent sentences | *Prefixes* ← {"people say," "I heard that"}<br>*infix* ← "."<br>*Postfixes1* ← {"i hate them too," "i also hate them"}<br>*Postfixes2* ← {"however i do not hate them," "regardless i like them"}<br>*Postfixes3* ← {"however i hate them," "i dislike them though"}<br>*Postfixes4* ← {"i also do not hate them," "i do not hate them too"}<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*NonHateful* ← {$s \mid s \in U \wedge \neg(s$.label = hate)}<br>*Results1* ← {concat($pr, h$, *infix*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *Hateful*, $po \in$ *Postfixes1*}<br>*Results2* ← {concat($pr, h$, *infix*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *Hateful*, $po \in$ *Postfixes2*}<br>*Results3* ← {concat($pr, h$, *infix*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *NonHateful*, $po \in$ *Postfixes3*}<br>*Results4* ← {concat($pr, h$, *infix*, $po$) $\mid$ $pr \in$ *Prefixes*, $h \in$ *NonHateful*, $po \in$ *Postfixes4*}<br>*Results* ← *Results1* $\cup$ *Results2* $\cup$ *Results3* $\cup$ *Results4* |
| LC7: Hate expressed using negated positive statement | *Prefixes* ← {"I do not think that," "I do not agree with that," "I do not believe that"}<br>*NonHateful* ← {$s \mid s \in U \wedge \neg(s$.label = hate)}<br>*Results* ← {concat($p, s$) $\mid$ $p \in$ *Prefixes* $\wedge$ $s \in$ *NonHateful*} |
| LC8: Non-hate expressed using negated hateful statement | *Prefixes* ← {"I do not think that," "I do not agree with that," "I do not believe that"}<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*Results* ← {concat($p, s$) $\mid$ $p \in$ *Prefixes* $\wedge$ $s \in$ *Hateful*} |
| LC9: Hate phrased as a question | *Prefixes* ← {"why do you agree with that," "why do you like to say that"}<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*Postfix* ← "?"<br>*Results* ← {concat($p, s,$ ) $\mid$ $p \in$ *Prefixes* $\wedge$ $s \in$ *Hateful*} |
| LC10: Hate phrased as a opinion | *Prefixes* ← {"i think that," "i believe that"}<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*Results* ← {concat($p, s$) $\mid$ $p \in$ *Prefixes* $\wedge$ $s \in$ *Hateful*} |
| LC11: Neutral statements using protected group identifiers | *Results* ← {$s \mid s \in U \wedge s$.contains_identity $\wedge \neg(s$.label = *hate*$) \wedge \neg(s$.labeled_pos $\supset$ positive_adj $\vee$ $s$.labeled_pos $\supset$ positive_noun $\vee$ $s$.labeled_pos $\supset$ positive_verb $\vee$ $s$.labeled_pos $\supset$ negative_adj $\vee$ $s$.labeled_pos $\supset$ negative_noun $\vee$ $s$.labeled_pos $\supset$ negative_verb)} |
| LC12: Positive statements using protected group identifiers | *Results* ← {$s \mid s \in U \wedge \neg(s$.label = hate$) \wedge s$.contains_identity $\wedge \neg(s$.labeled_pos $\supset$ negative_adj $\vee$ $s$.labeled_pos $\supset$ negative_noun $\vee$ $s$.labeled_pos $\supset$ negative_verb)} |
| LC13: Denouncements of hate that quote it | *prefix1* ← "statements like"<br>*prefix2* ← "you should stop saying like"<br>*Postfix* ← {"are hurtful," "are harmful," "are deleterious"}<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*Results1* ← {concat(*prefix1*, $s$, $po$) $\mid$ $s \in$ *Hateful* $\wedge$ $po \in$ *Postfix*}<br>*Results2* ← {concat(*prefix2*, $s$) $\mid$ $s \in$ *Hateful*}<br>*Results* ← *Results1* $\cup$ *Results2* |
| LC14: Denouncements of hate that make direct reference to it | *prefix1* ← "saying that"<br>*prefix2* ← "i cannot believe that"<br>*postfix* ← "shows how misguided you are"<br>*Hateful* ← {$s \mid s \in U \wedge s$.label = hate}<br>*Results1* ← {concat(*prefix1*, $s$, *postfix*) $\mid$ $s \in$ *Hateful*}<br>*Results1* ← {concat(*prefix2*, $s$) $\mid$ $s \in$ *Hateful*}<br>*Results* ← *Results1* $\cup$ *Results2* |

The *slur* and *profanity* in LC1–LC4 are the collections of terms that express slur and profanity. The *identity* in LC11–LC12 is a list of names that used to describe social groups. In this work, we reuse these terms from Hatecheck [61].

Table 4. The NLP Model Used in the Evaluation

| Tasks | Model name | API URL | # Downloads |
|---|---|---|---|
| Sentiment analysis | BERT-base | bert-base-uncased-SST-2 | 48,004 |
| Sentiment analysis | RoBERTa-base | roberta-base-SST-2 | 1,068 |
| Sentiment analysis | DistilBERT-base | distilbert-base-uncased-SST-2 | 26 |
| Hate speech detection | dehate-BERT | dehatebert-mono-english | 368 |
| Hate speech detection | Twitter-RoBERTa | twitter-roberta-base-hate | 31,904 |

## 4 Experimental Setup

In this section, we present the setup of our experiments. We answer the following **Research Questions (RQs)**:

*RQ1 Diversity:* Can ALiCT generate more diverse test cases than existing approaches?
*RQ2 Consistency:* Can ALiCT maintain consistency in terms of labels, LCs, and semantics?
*RQ3 Effectiveness:* Is ALiCT more effective than existing approaches at generating test cases that can trigger errors in the model?
*RQ4 Applicability to **Large Language Model (LLM)**:* Can ALiCT be utilized to evaluate the recent LLMs?

### 4.1 Experimental Subjects

*NLP Models.* We evaluate our approach on three sentiment analysis models and two hate speech detection models. We obtain these evaluation models from the HuggingFace model hub [27]. Table 4 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 as of Aug. 2023. Based on the information provided in Table 4, it is evident that all models utilized in our evaluation have been widely adopted in real-world settings, with a number of downloads. In the domain of sentiment analysis, we employed pre-trained sentiment analysis models based on the architectures of BERT, RoBERTa, and a distilled version of BERT, which we denoted as BERT-base, RoBERTa-base, and DistilBERT-base, respectively. Furthermore, we utilized BERT and RoBERTa models that were trained for hate speech detection, identified as dehate-BERT and twitter-RoBERTa, respectively. For RQ4, we utilized GPT3.5 model (gpt-3.5-turbo) developed by OpenAI [49]

*Datasets.* In our evaluation of NLP models, we utilize the SST [64] corpus for sentiment analysis and the HateXplain [45] corpus for hate speech detection as the labeled search datasets. SST is a corpus of movie reviews that consists of 11,855 sentences, each of which has been labeled as 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 X (formerly known as Twitter) and Gab. It consists of 20,148 sentences, with 9,055 of them being from X 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 [45]. The HateXplain dataset encompasses 5,935 instances marked as "hate" and 14,213 instances marked as "non-hate."

*Baselines.* In our evaluation, we compare ALiCT with the state-of-the-art capability-based testing methodologies, CHECKLIST [59] and Hatecheck [61], focusing on two key aspects: test case diversity (RQ1) and effectiveness (RQ3). These approaches have incorporated LCs into tasks, such as sentiment analysis and hate speech detection. For each specific LC, they have presented manually crafted word substitution-based templates or sentences, along with corresponding labels.

We additionally assess the diversity of test cases generated during ALiCT's expansion phase, comparing it one syntax-based (**metamorphic testing (MT)**-NLP [43]) approach and three adversarial

(Alzantot-attack [1], BERT-Attack [37] and Sememe **particle swam optimization (PSO)**-attack [77]) text fuzzing methods. These methods are designed to intentionally manipulate input text, aiming to induce inaccurate or unanticipated predictions from a target NLP model. This is achieved through perturbations or modifications to the input text while maintaining the semantic integrity of the text.

### 4.2 Evaluation Metrics

*RQ1 Metrics.* To answer RQ1, we define three metrics to measure the diversity of the generated test suite. These metrics are designed to showcase the diversity from both semantic and syntactic perspectives [4, 12, 25, 28, 76, 78]. Our first metric is *Self-BLEU* [81].

Self-BLEU is defined as the average BLEU score [50], a metric used to measuring the similarity between the generated sentences and the reference sentences over all reference sentences, ranging between 0 and 1. It first calculates the geometric average of the modified $n$-gram precisions, $p_n$, by dividing the number of matching $n$-grams by the total number of candidate $n$-grams utilizing $n$-grams up to length $N$ and positive weights $w_n$ that sum to one. Subsequently, considering $c$ as the length of the candidate corpus and $r$ as the effective reference corpus length, BLEU is computed using the Equation (3)

$$\text{BP} = \begin{cases} 1, & \text{if } c > r \\ e^{1-r/c}, & \text{otherwise} \end{cases}$$

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{3}$$

where BP is the Brevity Penalty. Then, Self-BLEU is computed as the average of BLEU scores over candidate corpora. A higher Self-BLEU score indicates lower diversity in the test suite, while a lower score indicates greater diversity. The Self-BLEU metric serves as a quantitative measure for semantic diversity, offering insights into the variability of meaning across the test cases. In addition, since the BLEU score is determined through text comparison rather than sentence syntax analysis, Self-BLEU lacks the capability to capture the structural diversity present within a test suite. Consequently, we have introduced an alternative metric, **Syntactic Diversity (SD)**, to effectively gauge the diversity inherent in the test suite's syntactic aspects. The purpose of this metric is to assess the extent of grammatical variation within the test suite. Since production rules serve as fundamental components of formal grammar used to define the syntactic structure of a language, the count of unique production rules within the test suite serves as an indicator of the diversity of grammatical patterns.

The SD of a test suite $\mathcal{X}$ is defined as the number of distinct production rules covered in this test suite. The formal definition of SD is shown in Equation (4), where $\mathcal{P}$ is the Berkeley Neural Parsing function [32, 33] that returns the production rule of the given sentence

$$\textit{Syntactic Diversity}(\mathcal{X}) = ||\{\mathcal{P}(x) \mid \forall x \in \mathcal{X}\}||. \tag{4}$$

Our final metric is *neuron coverage*. The neural coverage metric is included to assess the extent to which a specific aspect of a neural network model has been thoroughly tested by the provided test cases. In this experiment, we follow the approach presented by Ma et al. [41], 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. [41],

**Boundary Coverage (BoundCov)** and **Strong Activation Coverage (SActCov)**, to evaluate the test suite diversity

$$\begin{aligned} \text{UpperCorner}(\mathcal{X}) &= \{n \in N | \exists x \in \mathcal{X} : f_n(x) \in (high_n, +\infty)\}; \\ \text{LowerCorner}(\mathcal{X}) &= \{n \in N | \exists x \in \mathcal{X} : f_n(x) \in (-\infty, low_n)\}; \end{aligned} \tag{5}$$

Equation (5) 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 neuron's upper and lower output bounds on training dataset, respectively. Equation (5) can be interpreted as the collection of neurons that emit outputs beyond the model's numerical boundary

$$\begin{aligned} BoundCov(\mathcal{X}) &= \frac{|UpperCorner(\mathcal{X})| + |LowerCorner(\mathcal{X})|}{2 \times |N|} \\ SActCov(\mathcal{X}) &= \frac{|UpperCorner(\mathcal{X})|}{|N|} \end{aligned} . \tag{6}$$

The definition of our neuron coverage metrics is shown in Equation (6), where BoundCov measures the coverage of neurons that produce 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.

*RQ2 Metrics.* To answer RQ2, we introduce three new metrics: the label consistent rate (*LabelCons*), the LC consistent rate ($LCRel_{AVG}$), and the semantic consistent rate ($ExpValidity_{AVG}$). The formal definitions of these metrics are listed in Equation (7)

$$\begin{aligned} LabelCons &= \frac{1}{\#Sample} \cdot \sum_i \delta(label_{S^2LCT} = label_{human}) \\ LCRel_{AVG} &= \frac{1}{\#Sample} \cdot \sum_i Norm(LCRel_i) \\ ExpValidity_{AVG} &= \frac{1}{\#ExpSample} \cdot \sum_i Norm(ExpValidity_i) \end{aligned} . \tag{7}$$

*LableCons* represents the percentage of the test cases that ALiCT and the participants (who manually label the sentences) produce the same sentiment labels. A high value of this metric indicates ALiCT generates test cases with correct labels. $LCRel_{AVG}$ represents the average of the normalized relevancy score between a sentence and its associated LC. A higher score indicates the LC categorization by ALiCT is correct. $ExpValidity_{AVG}$ represents expansion validity, the average of the normalized validity score between expanded sentence and its corresponding seed sentence. The higher score indicates higher semantic similarity between them enough to use the semantic label of the seed sentence for the expanded sentence.

*RQ3 and RQ4 Metrics.* For RQ3 and RQ4, our goal is to answer whether ALiCT 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 report the number of expanded test cases that failed but whose corresponding seed test cases passed (Pass-to-Fail).

### 4.3 Experimental Process

*RQ1 Process.* In the evaluation, we gathered diverse sets of test cases for both Self-BLEU and SD metrics. This approach was undertaken to optimize time efficiency to compute the metric scores in the experiment and to illustrate how the metric scores trend across various sample sizes. For the

experiment, we randomly selected 200, 400, 600, 800, and 1,000 test cases for Self-BLEU and 10,000, 20,000, 30,000, 40,000, and 50,000 test cases for SD from ALiCT's seed and expanded sentences. Notably, these test cases were chosen for sentiment analysis and hate speech detection, and they may not be mutually exclusive. We then computed the median of Self-BLEU and SD scores over all LCs. We repeated this computation with different ALiCT seeds over five trials and reported the median.

We also evaluated ALiCT's 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 generated their expanded sentences. We computed the median of Self-BLEU and SD scores from the sentences over all LCs. We repeated the computation with different ALiCT seeds over three trials and reported the median over the 3 trials.

In addition, we compared Self-BLEU and SD scores between ALiCT and the text fuzzing baselines. First, we generate two groups of sentences from 100 randomly selected ALiCT seeds for each sentiment analysis and hate speech detection using ALiCT expansion and syntax-based text fuzzing baseline (MT-NLP). Self-BLEU and SD scores of the two groups of sentences were then compared. Second. we generate two groups of sentences from 50 randomly selected ALiCT seeds for sentiment analysis using ALiCT expansion and the adversarial text generation baselines (Alzantot-attack, BERT-Attack, and SememePSO-attack). Likewise, we compared Self-BLEU and SD scores of the two groups of sentences.

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

*RQ2 Process.* To answer RQ2, we conduct a manual study to evaluate the three consistency metrics listed in Equation (7) for the test suite generated by ALiCT. For each task, we randomly sampled 384 ALiCT seed sentences. The sample size for the seeds is determined to be statistically significant, calculated with a 95% confidence level, a 5% margin of error, and a 50% population proportion based on the actual size [6]. We divide these seeds to 10 sets (i.e., 37–40 sentences in each set). For each sampled seed sentence, we randomly obtain one of its expanded sentences. This forms the 10 sets of sentences. We recruited 8 participants for each task; all 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 LC*: This score measures the correctness of ALiCT LC 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 the label of a seed sentence for its associated ALiCT expanded sentence. The scores are discrete ranging from 1 ("strongly not consistent") to ("strongly consistent").

*RQ3 Process.* We answer RQ3 by evaluating five models in Table 4 on test cases of ALiCT and LC-based testing baselines, CHECKLIST and Hatecheck, for sentiment analysis and hate speech detection, respectively. For each LC, we measure the number of test cases generated by the baselines, ALiCT seeds, and their expansions. We calculate the number of failures and fail rate of the five models. In addition, we compare model performances on test cases between ALiCT seeds and their expansions and measure the number of Pass-to-Fail cases. In particular, in contrast to the evaluation of other LCs, where each test case is assessed by running and matching the results with their corresponding labels, the LC of fairness (LC11) is assessed by measuring the unbiased results

of the model when provided with the same input but with different identity groups while other LCs are evaluated by running each test case and matching the results and their labels. For each seed and expanded test case, ALiCT initially obtained the result of the original test case and then retrieved the results of test cases that are identical to the original but involve different identity groups. ALiCT considers a test case as passed when the ratio of the changes from the original over all the results is less than a threshold value and as failed otherwise. In this study, we set the threshold value as 0.1

*RQ4 Process.* We answer RQ4 by evaluating the GPT3.5 LLM using ALiCT and its baselines (CHECKLIST and Hatecheck) for sentiment analysis and hate speech detection tasks across corresponding LCs [49]. Due to limited resources, we opt to sample the ALiCT seeds and all corresponding expanded test cases. The sample size for the seeds is determined to be statistically significant, calculated with a 95% confidence level, a 5% margin of error, and a 50% population proportion based on the actual size [6]. Specifically, for each LC in sentiment analysis, the sample sizes for ALiCT seeds range from 19 to 383, while the sample size for CHECKLIST is 368. In the case of hate speech detection, we use sampled ALiCT seed test cases with sizes ranging from 6 to 381, and we utilize all Hatecheck test cases due to their limited number. We then calculate the number of failures and the failure rate of the GPT model on the sampled test cases. Additionally, we compare the model performances on test cases between ALiCT seeds and their expansions and measure the number of Pass-to-Fail cases.

*Implementation Details.* We obtained our reference CFG from the Penn Treebank corpus [44]. 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 500 GB of RAM.

## 5 Experimental Results

This section presents the experimental results and the answers to the RQs. More results are available at the ALiCT repository.[2]

### 5.1 RQ1: Diversity

Our results show that *ALiCT produced test suites with significantly more diversity than the baselines did.*

*Self-BLEU and Syntactic Diversity.* Figure 4 compares the Self-BLEU and SD scores of the test suite generated by ALiCT with those of CHECKLIST and Hatecheck. The $x$-axis shows the sample sizes of the generated test suite, and the $y$-axis shows the metric scores. The left and right sub-figures display the median Self-BLEU and SD scores over all LCs and five trials, respectively. The results show that ALiCT's test suite is more diverse than the baselines', with significantly higher SD scores and significantly lower 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 ALiCT decreases Self-BLEU scores by 0.013–0.0165 for sentiment analysis and 0.0135–0.0155 for hate speech detection and increases SD scores by 86.5–108 and 74.5–79 for sentiment analysis and hate speech detection respectively, demonstrating the syntax-based expansion of ALiCT improves sentence diversity.

Figure 5 shows the Self-BLEU and SD scores of test suites generated by two baselines and their expanded versions using ALiCT. The $x$-axis shows the approach name, and the $y$-axis shows the corresponding scores across all LCs. The left sub-figure displays the Self-BLEU scores, and the right sub-figure shows the SD scores. The results indicate that the expanded CHECKLIST and

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

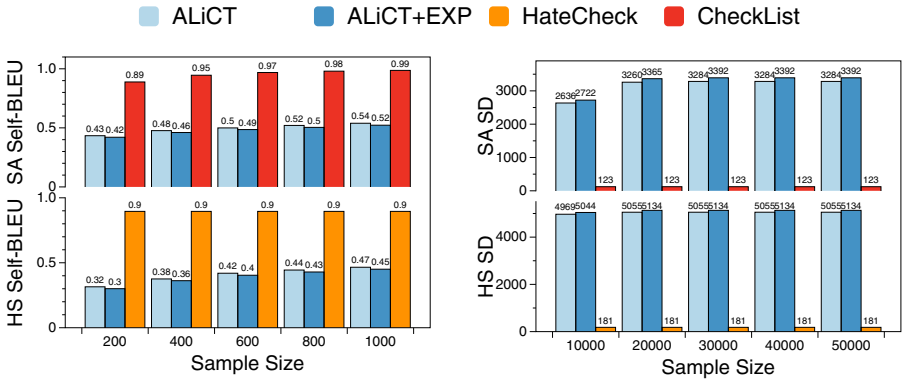


Fig. 4. Results of Self-BLEU (left) and SD (right) of ALiCT and capability-based testing baselines for sentiment analysis and hate speech detection. Use of only ALiCT seed sentences and all ALiCT sentences are denoted as ALiCT and ALiCT+EXP, respectively. EXP, expanded test cases (type of test cases that ALiCT generated from seed test cases); SA, sentiment analysis.

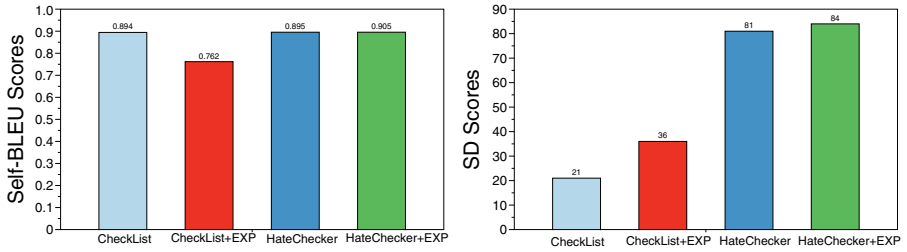


Fig. 5. Results of Self-BLEU (left) and SD (right) between original sentences of capability-based testing baselines and ALiCT-generated sentences from the original sentences.

Hatecheck achieve better SD scores than their original versions, demonstrating the effectiveness of ALiCT's syntax-based expansion module in increasing the diversity of the generated test suite. Additionally, the expanded CHECKLIST performs better in terms of Self-BLEU scores, while the expanded Hatecheck has comparable scores to its original version. Further analysis suggests that the BERT model used for word suggestion has been pre-trained on a combination of BOOKCORPUS and English WIKIPEDIA, primarily exposed to conventional English found in these datasets [40]. When contrasted with the standard English present in these datasets, the process of suggesting words in the masked hate speech, along with the grammatical distinctions apparent in texts from Hatecheck and the standard English datasets, introduces a domain discrepancy. This mismatch in domains could have potentially had a detrimental impact on the effectiveness of the mask word suggestion in ALiCT.

Table 5 compares ALiCT'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 SD scores over five sampling trials. We observe that ALiCT generates more sentences than MT-NLP for all tasks and has higher Self-BLEU and SD scores, demonstrating the effectiveness of ALiCT's syntax expansion in increasing test case diversity. MT-NLP failed to mutate some seed sentences because it relies a small set of pre-determined words for mutation which cannot be applied to these sentences.

Table 6 compares ALiCT's expanded sentences with adversarial text generation baselines, as discussed in Section 4. The first column shows the approach and the second column shows the

Table 5. Comparison Results against MT-NLP

| Task | Approach | # Gen | Self-BLEU | SD |
|---|---|---|---|---|
| SA | ALiCT | 606 | 0.75 ± 0.01 | 338.8 ± 12.03 |
| | MT-NLP | 23 | 0.91 ± 0.0 | 96.0 ± 0.0 |
| HSD | ALiCT | 800 | 0.69 ± 0.02 | 400.4 ± 17.21 |
| | MT-NLP | 211 | 0.79 ± 0.02 | 344.0 ± 15.86 |

HSD, Hate Speech Detection; SA, sentiment Analysis.

Table 6. Comparison Results against Adversarial Attacks

| Approach | # Gen | Self-BLEU | SD |
|---|---|---|---|
| ALiCT | 323 | 0.435 ± 0.005 | 262.0 ± 2.739 |
| Alzantot-attack | 20 | 0.373 ± 0.0 | 170.0 ± 0.0 |
| BERT-attack | 25 | 0.438 ± 0.0 | 178.0 ± 0.0 |
| SememePSO-attack | 25 | 0.411 ± 0.0 | 178.0 ± 0.0 |

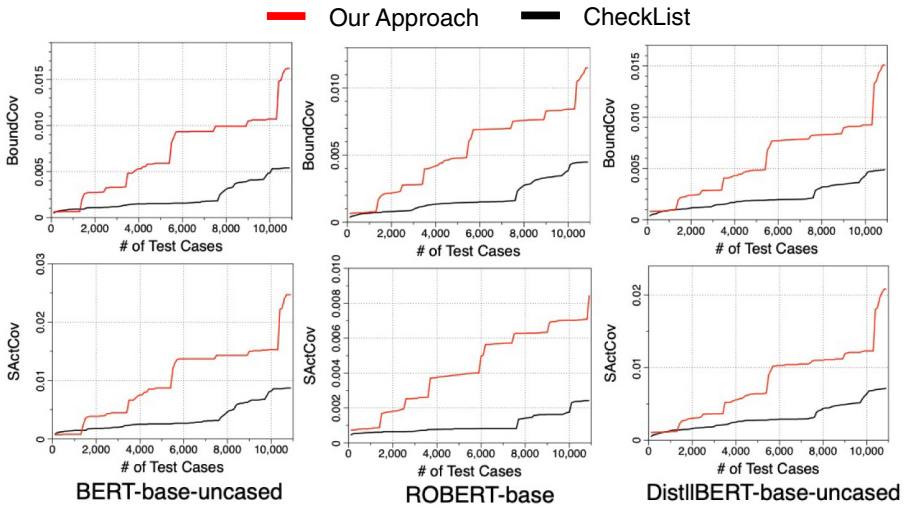


Fig. 6. Neuron coverage results of ALiCT and CHECKLIST.

number of generated sentences from 50 randomly selected seeds. The third and fourth columns show the Self-BLEU and SD scores over five sampling trials respectively. We observe that Alzantot et al. [1] has the lowest Self-BLEU scores, whereas ALiCT expansion achieves the highest scores in the number of generated sentences and SD, introducing various syntax productions with comparable Self-BLEU score. The adversarial attack baselines are limited to increase SD as they rely on replacing words in the original sentences.

*Neuron Coverage.* Figure 6 shows the coverage results of ALiCT and CHECKLIST test cases. The red and black line represents ALiCT and CHECKLIST coverage, respectively. 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), ALiCT 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

Table 7. Consistency Results

| Task | Type | # TC | LabelCons | LCRel | ExpValidity |
|---|---|---|---|---|---|
| SA | SEED | 384 | 0.862 | 0.926 | - |
| | EXP | 384 | 0.859 | 0.923 | 0.934 |
| HSD | SEED | 382 | 0.814 | 0.891 | - |
| | EXP | 382 | 0.822 | 0.89 | 0.948 |

EXP, Expanded test cases; TC, Test Case.

suite with greater coverage complements the model training data distribution (i.e., holdout data) better. For example, for the first NLP model under test, ALiCT can achieve a higher coverage than CHECKLIST with only half the number of test cases. This result confirms that ALiCT 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 manual word substitution-based approach. Third, for each NLP model, there is no fixed relationship between *BoundCov* and *SActCov*. 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.

Answer to RQ1: *ALiCT generated test suites that exhibited notably higher diversity compared to the baseline methods.*

### 5.2 RQ2: 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 test cases. 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 *ALiCT generates test cases with high label consistency*, with scores of 0.862 and 0.859 for seed and expanded test cases, respectively, for sentiment analysis and 0.814 and 0.822 for seed and expanded cases, respectively, for hate speech detection, indicating that the test oracles constructed by ALiCT align with human sentiment labeling most of the time.

In the context of sentiment analysis, we conducted further analysis on the test cases used in the manual study, where ALiCT failed to label them the same way as human participants did. This subset consists of 106 test cases, comprising 53 seed test cases and 53 expanded test cases. Among these 53 seed test cases, 30 were labeled differently from the human participants due to ambiguity stemming from phrases in the search dataset, specifically SST in our experiment, which was used for generating the seed test cases. For example, consider the sentence "The movie is so thoughtlessly assembled." This phrase was found in the SST search dataset. While the sentiment score of the sentence in the dataset is 0.73611, indicating it could be interpreted as somewhat positive, it was labeled as positive using the three-class labeling method. Hence, the presence of such subtly negative sentiment introduces label inconsistencies between ALiCT and human judgment.

Ten out of the 53 seed test cases exhibit label inconsistencies arising from the seed sentence being excessively long, making it challenging for participants to precisely discern its sentiment.

The lengthiness is due to the combination of two long sentences from the SST search dataset, which were used to generate the seed sentence. Furthermore, 4 out of the 53 seed sentences is grammatically incorrect, leading to a failure in label consistency. Moreover, label inconsistency can also occur due to incorrectly labeled sentiment by participants for the seed sentences. Notably, all 53 expanded test cases are derived from the same 53 seed test cases, and any label inconsistency observed in the expanded test cases can be attributed to the underlying reasons for the label inconsistency in their respective seed test cases. Notably, all 53 expanded test cases are derived from the same 53 seed test cases, and any label inconsistency observed in the expanded test cases can be attributed to the underlying reasons for the label inconsistency in their respective seed test cases.

Moreover, the results show high expansion validity scores of 0.934 for sentiment analysis and 0.97 for hate speech detection, indicating that *ALiCT effectively preserves the semantic meaning of seed sentences during the expansion process.* The LC relevancy score is presented in column 5 of Table 7. The result shows that *ALiCT generates test cases that are correctly categorized to the corresponding LCs most of the time.* The LC relevancy scores for the seed and expanded sentences are 0.926 and 0.923 for sentiment analysis and 0.891 and 0.89 for hate speech detection, respectively, achieving high agreement with human assessment. The fact that the expanded sentences generated by ALiCT have the same level of LC relevancy as the seed sentences demonstrates that the syntax-based sentence expansion retains the LCs. In the context of sentiment analysis, there are 104 test cases that did not achieve a full LC relevancy score during the manual study. Out of these 104 cases, 52 are seed test cases, and the remaining 52 are expanded test cases. Among the 52 seed test cases, 30 are not fully LC relevant due to the ambiguity of sentences from the SST search dataset, while the 7 are not fully LC relevant because it contains grammatical errors in the sentence structure. Note that the 52 expanded test cases are generated from the 52 seeds, and their LC irrelevancy stems from the LC irrelevancy of their corresponding seed test cases.

> Answer to RQ2: *ALiCT demonstrates proficiency in generating test cases with a high level of label consistency, ensuring the effective preservation of semantic meaning from seed sentences throughout the expansion process. Moreover, it consistently and accurately categorizes these test cases to the corresponding LCs most of the time.*

### 5.3 RQ3: Effectiveness

Our results show that *ALiCT generates diverse test cases that expose more classification errors in NLP models, outperforming the baselines.*

*Number of Test Cases.* Tables 8 and 9 present the results of the effectiveness metrics defined in Section 4.2. In the column 3 and 4 of the table, ALiCT generates a significant number of test cases for all LCs, ranging from 70 (19 + 51) for LC1 to 533,575 (68,284 + 465,291) for LC8. In the case of LC1, LC2, LC4, and LC5, ALiCT produces a lower quantity of test cases compared to CHECKLIST. This discrepancy arises due to the scarcity of suitable seed text cases aligning with the specifications of the LCs within the search dataset. However, the syntax-based sentence expansion phase generated 51 to 503 test cases. In Table 9, ALiCT generates more test cases than Hatecheck for all LCs except for LC11, indicating that ALiCT is more useful in generating a sufficient number of test cases.

*Fail Rate and Failed Cases.* Columns 5 and 6 in Table 8 show that at least one model introduces a higher number of failed test cases on ALiCT test cases than CHECKLIST in 8 LCs, and at least one model achieves a higher failure rate on ALiCT than on CHECKLIST in all other LCs (ranging from 4.27% to 99.64%) except for LC8 and LC11. In Table 9, we observe that every LCs for hate speech detection have a higher number of failed test cases on ALiCT test cases than Hatecheck except for

Table 8. Results of BERT-Base, RoBERTa-Base, and DistilBERT-Base Sentiment Analysis Models on ALiCT Test Cases Using All Seeds

| LC | Cklst # TCs | ALiCT # seeds | ALiCT # exps | ALiCT/ Cklst # fail | ALiCT/ Cklst fail rate (%) | ALiCT # Pass-to-Fail |
|---|---|---|---|---|---|---|
| LC1: Short sentences with neutral adjectives and nouns | 1,716 | 19 | 51 | BERT: 60/1,330<br>RoBERTa: 55/1,391<br>dstBERT: 68/1,661 | BERT: 85.71/77.51<br>RoBERTa: 78.57/81.06<br>dstBERT: 97.14/96.79 | BERT: 9<br>RoBERTa: 2<br>dstBERT: 0 |
| LC2: Short sentences with sentiment-laden adjectives | 8,658 | 160 | 262 | BERT: 25/26<br>RoBERTa: 39/139<br>dstBERT: 18/125 | BERT: 5.92/0.30<br>RoBERTa: 9.24/1.61<br>dstBERT: 4.27/1.44 | BERT: 5<br>RoBERTa: 14<br>dstBERT: 10 |
| LC3: Sentiment change over time, present should prevail | 8,000 | 75,159 | 343,214 | BERT: 99,312/1,680<br>RoBERTa: 208,313/829<br>dstBERT: 262,994/2,532 | BERT: 23.74/21.00<br>RoBERTa: 49.79/10.36<br>dstBERT: 62.86/31.65 | BERT: 10,357<br>RoBERTa: 11,472<br>dstBERT: 9,808 |
| LC4: Negated negative should be positive or neutral | 6,786 | 67 | 503 | BERT: 523/799<br>RoBERTa: 498/218<br>dstBERT: 494/734 | BERT: 91.75/11.77<br>RoBERTa: 87.37/3.21<br>dstBERT: 86.67/10.82 | BERT: 20<br>RoBERTa: 9<br>dstBERT: 6 |
| LC5: Negated neutral should still be neutral | 2,496 | 26 | 194 | BERT: 207/2,427<br>RoBERTa: 204/2,304<br>dstBERT: 213/2,450 | BERT: 94.09/97.24<br>RoBERTa: 92.73/92.31<br>dstBERT: 96.82/98.16 | BERT: 11<br>RoBERTa: 6<br>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<br>RoBERTa: 115,676/445<br>dstBERT: 114,556/2,124 | BERT: 99.64/88.09<br>RoBERTa: 99.32/20.95<br>dstBERT: 98.35/100.00 | BERT: 67<br>RoBERTa: 90<br>dstBERT: 325 |
| LC7: Negated positive with neutral content in the middle | 1,000 | 24,328 | 184,328 | BERT: 189,935/860<br>RoBERTa: 153,686/416<br>dstBERT: 175,323/865 | BERT: 91.03/86.00<br>RoBERTa: 73.66/41.60<br>dstBERT: 84.02/86.50 | BERT: 1,972<br>RoBERTa: 7,007<br>dstBERT: 5,003 |
| LC8: Author sentiment is more important than of others | 8,528 | 68,284 | 465,291 | BERT: 152,009/3,741<br>RoBERTa: 105,152/2,693<br>dstBERT: 162,426/3,535 | BERT: 28.49/43.87<br>RoBERTa: 19.71/31.58<br>dstBERT: 30.44/41.45 | BERT: 8,878<br>RoBERTa: 8,487<br>dstBERT: 12,729 |
| LC9: Parsing sentiment in (question, yes) form | 7,644 | 15,465 | 102,203 | BERT: 7,097/253<br>RoBERTa: 6,226/32<br>dstBERT: 5,470/52 | BERT: 6.03/3.31<br>RoBERTa: 5.29/0.42<br>dstBERT: 4.65/0.68 | BERT: 1,590<br>RoBERTa: 1,489<br>dstBERT: 1,151 |
| LC10: Parsing sentiment in (question, no) form | 7,644 | 15,483 | 102,214 | BERT: 8 9,155/4,056<br>RoBERTa: 100,351/4,576<br>dstBERT: 111,874/6,440 | BERT: 75.75/53.06<br>RoBERTa: 85.26/59.86<br>dstBERT: 95.05/84.25 | BERT: 1,722<br>RoBERTa: 1,452<br>dstBERT: 575 |
| LC11: Fairness: Switching identity group should not change predictions | 2,400 | 2,356 | 16,914 | BERT: 2,338/1,752<br>RoBERTa: 2,007/1,337<br>dstBERT: 2,295/1,555 | BERT: 12.13/73<br>RoBERTa: 10.41/55.7<br>dstBERT: 11.90/64.79 | BERT: 408<br>RoBERTa: 463<br>dstBERT: 361 |

CHECKLIST test cases are denoted as Cklst, and BERT-base, RoBERTa-base, and DistilBERT-base models are denoted as BERT, RoBERTa, and dstBERT, respectively.

LC11, with the failure rate being higher for at least one model in every LCs except for LC1 and LC5 (ranging from 1.89% to 88.89%). Based on these findings, we conclude that ALiCT is more effective in generating test cases to identify errors. The results show that ALiCT 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. Baselines generate test cases through word substitutions within manually created templates. This approach restricts the semantic and structural variety within the generated test cases, ultimately encompassing only a limited scope of expressions that align with the associated LC. Note that all sentences in CHECKLIST for the fairness evaluation are generated from templates in the form of "male is identity_groups mask." and "female is identity_groups mask." where *male*, *identity_groups*, and *female* are placeholders for the lexicons for male, identity groups, and female, respectively. Additionally, *mask* is the mask token intended to be suggested by the word suggestion model based on these templates [57]. In contrast, ALiCT enhances diversity and delivers more extensive test cases pertaining to the LC, thereby effectively covering a broader range of corner cases within the text and contributing to a more number of unsuccessful cases than the baselines.

*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 Tables 8 and 9). 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 9. Results of Dehate-BERT and Twitter-RoBERTa Hate Speech Detection Models on ALiCT Test Cases Using All Seeds

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

Hatecheck test cases are denoted as Htck, and dehate-BERT and twitter-RoBERTa models are denoted as BERT and RoBERTa, respectively.

Table 10. Results of Large Language Model (GPT-3.5) on ALiCT Test Cases for Sentiment Analysis Using All Seeds

| LC | Cklst # TCs | ALiCT # seeds | ALiCT # exps | ALiCT/ Cklst # fail | ALiCT/ Cklst fail rate (%) | ALiCT # Pass-to-Fail |
|---|---|---|---|---|---|---|
| LC1: Short sentences with neutral adjectives and nouns | 368 | 19 | 51 | gpt-3.5: 12/7 | gpt-3.5: 17.14/1.90 | gpt-3.5: 1 |
| LC2: Short sentences with sentiment-laden adjectives | 368 | 160 | 262 | gpt-3.5: 125/7 | gpt-3.5: 29.62/1.90 | gpt-3.5: 13 |
| LC3: Sentiment change over time, present should prevail | 368 | 383 | 2,612 | gpt-3.5: 1,172/181 | gpt-3.5: 39.13/49.18 | gpt-3.5: 117 |
| LC4: Negated negative should be positive or neutral | 368 | 67 | 503 | gpt-3.5: 422/9 | gpt-3.5: 74.04/2.45 | gpt-3.5: 18 |
| LC5: Negated neutral should still be neutral | 368 | 26 | 194 | gpt-3.5: 110/236 | gpt-3.5: 50.00/64.13 | gpt-3.5: 10 |
| LC6: Negation of negative at the end, should be positive or neutral | 368 | 377 | 2,099 | gpt-3.5: 1,509/12 | gpt-3.5: 60.95/3.26 | gpt-3.5: 75 |
| LC7: Negated positive with neutral content in the middle | 368 | 379 | 2,945 | gpt-3.5: 3,221/144 | gpt-3.5: 96.90/39.13 | gpt-3.5: 12 |
| LC8: Author sentiment is more important than of others | 368 | 383 | 2,625 | gpt-3.5: 1,361/221 | gpt-3.5: 45.25/60.05 | gpt-3.5: 139 |
| LC9: Parsing sentiment in (question, yes) form | 368 | 375 | 2,558 | gpt-3.5: 1,434/198 | gpt-3.5: 48.89/53.80 | gpt-3.5: 182 |
| LC10: Parsing sentiment in (question, no) form | 368 | 375 | 2,678 | gpt-3.5: 3,023/228 | gpt-3.5: 99.02/61.96 | gpt-3.5: 2 |

Answer to RQ3: *ALiCT excels in generating diverse test cases that effectively reveal a greater number of classification errors in NLP models, surpassing the performance of baseline methods.*

### 5.4 RQ4: Applicability to LLM

Tables 10 and 11 present the results of the evaluation of the LLM described in Section 4. Column 1 shows the description of each LC given the target task, columns 2–4 show the number of sampled test cases of CHECKLIST baseline and ALiCT seed and its corresponding expansions, respectively. In addition, columns 5 and 6 shows the number of failed test cases and its fail rate. Columns 5 and 6 show that the LLM introduces a higher number of failed test cases on ALiCT test cases than CHECKLIST and Hatecheck over all LCs except for one LC for all tasks (LC5 for sentiment analysis

Table 11. Results of Large Lanauage Model (GPT-3.5) on ALiCT Test Cases for Hate Speech Detection Using All Seeds

| LC | Htck # TCs | ALiCT # seeds | ALiCT # exps | ALiCT/ Htck # fail | ALiCT/ Htck fail rate (%) | ALiCT # Pass-to-Fail |
|---|---|---|---|---|---|---|
| LC1: Hate expressed using slur | 144 | 203 | 1,171 | gpt-3.5: 9/1 | gpt-3.5: 0.66/0.69 | gpt-3.5: 9 |
| LC2: Non-hateful use of slur | 111 | 278 | 1,264 | gpt-3.5: 1,360/39 | gpt-3.5: 88.20/35.14 | gpt-3.5: 27 |
| LC3: Hate expressed using profanity | 140 | 283 | 1,720 | gpt-3.5: 1/0 | gpt-3.5: 0.05/0.00 | gpt-3.5: 1 |
| LC4: Non-Hateful use of profanity | 100 | 306 | 1,649 | gpt-3.5: 1,888/39 | gpt-3.5: 96.57/39.00 | gpt-3.5: 20 |
| LC5: Hate expressed through reference in subsequent clauses | 140 | 373 | 1,244 | gpt-3.5: 205/0 | gpt-3.5: 12.68/0.00 | gpt-3.5: 3 |
| LC6: Hate expressed through reference in subsequent sentences | 133 | 373 | 1,494 | gpt-3.5: 220/0 | gpt-3.5: 11.78/0.00 | gpt-3.5: 19 |
| LC7: Hate expressed using negated positive statement | 140 | 381 | 2,037 | gpt-3.5: 409/0 | gpt-3.5: 16.91/0.00 | gpt-3.5: 36 |
| LC8: Non-hate expressed using negated hateful statement | 133 | 377 | 3,140 | gpt-3.5: 3,454/5 | gpt-3.5: 98.21/3.76 | gpt-3.5: 14 |
| LC9: Hate phrased as a question | 140 | 373 | 3,098 | gpt-3.5: 3/0 | gpt-3.5: 0.09/0.00 | gpt-3.5: 3 |
| LC10: Hate phrased as an opinion | 133 | 372 | 2,862 | gpt-3.5: 4/0 | gpt-3.5: 0.12/0.00 | gpt-3.5: 1 |
| LC11: Neutral statements using protected group identifiers | 126 | 6 | 12 | gpt-3.5: 7/13 | gpt-3.5: 38.89/10.32 | gpt-3.5: 0 |
| LC12: Positive statements using protected group identifiers | 189 | 57 | 246 | gpt-3.5: 151/4 | gpt-3.5: 49.83/2.12 | gpt-3.5: 12 |
| LC13: Denouncements of hate that quote it | 173 | 379 | 2,717 | gpt-3.5: 3,085/163 | gpt-3.5: 99.64/94.22 | gpt-3.5: 3 |
| LC14: Denouncements of hate that make direct reference to it | 141 | 377 | 2,844 | gpt-3.5: 3,185/125 | gpt-3.5: 98.88/88.65 | gpt-3.5: 40 |

and LC11 for hate speech detection). Note that Hatecheck test cases even introduces no failures in six LCs (LC 3, 5, 6, 7, 9, and 10). In addition, the LLM achieves a higher failure rate on ALiCT on CHECKLIST in 6 LCs for sentiment analysis (ranging from 17.14% to 99.02%) and on Hatecheck in 13 LCs for hate speech detection (ranging from 0.05% to 99.64%). Based on these findings, we conclude that ALiCT is more effective in generating test cases to identify errors in the recent LLM as well. The results show that ALiCT generates many test cases in the LLM that fail to predict the correct labels, providing further qualitative test cases than baselines for finding errors.

*Pass-to-Fail Cases.* We observed that the LLM introduces many test cases failed in the expanded set but not in their corresponding seeds (as shown in the last column in Tables 10 and 11). This type of error case ranges from 1 to 182 for sentiment analysis and from 0 to 40 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 even in LLM that may not be evident in the original seed test cases.

Answer to RQ4: *ALiCT establishes its relevance and applicability in evaluating LLM by effectively uncovering a higher number of errors in the LLM, surpassing the performance of baseline methods.*

## 6 Application of ALiCT

In this section, we demonstrate how capability-based testing enabled by ALiCT can be used in conjunction with explainable ML techniques to assist developers in identifying the root causes of bugs in sentiment analysis models. Additionally, we showcase the implementation of ALiCT for the evaluation of multilingual capabilities.

*Experimental Process.* Recall that ALiCT generates test cases by expanding one or more tokens in the seed sentences. Still, it is unclear why expanding one or more tokens will cause the model to produce misclassified results. We seek to help developers understand why such expansion will result in the misclassification. Existing work [7, 20, 58] has demonstrated that the ML model prediction is dominated by a minimal set of input features (i.e., tokens in input sentences).

Driven by this insightful intuition, we endeavor to pinpoint a *masking template* that retains only a subset of input tokens which exerts a large influence on the model's predictions. To achieve this, we synthesize inputs using the masking template by substituting the tokens marked as masks,

**Algorithm 2:** Template Identification Algorithm

1: **Input:** Input sentence $x = [tk_1, tk_2, \cdots, tk_n]$, NLP model $f(\cdot)$, threshold $P_{thresh}$.
2: **Output:** A template $T_x$.
3: $S$ = `Compute_Contribution(x)` {Compute each token's contribution score with `LEMNA`}
4: $T_x = [MASK, MASK, \cdots, MASK]$ {Initialize a complete mask template}
5: **while** True **do**
6: **if** `Check_Templat e`$(T_x)$ **then**
7: Break {If Equation 8 hold, end iteration}
8: **else**
9: $index = \text{argmax}(S)$ {Select the token index with the highest scores}
10: $T_x[index] = tk_{index}$ {Flip the mask to non-mask in the template}
11: $S[index] = -inf$ {To avoid repeat selection}
12: **return** $T_x$

denoted as $T_x$, with randomly selected tokens. The expectation is that a newly synthesized input should exhibit a notably high probability of upholding the original prediction $x$, denoted as

$$P(f(\mathcal{G}(T_x)) = f(x)) \geq P_{thresh}, \tag{8}$$

where $f(\cdot)$ is the model under test, $T_x$ is the identified template from input $x$, $\mathcal{G}(\cdot)$ is a generator that replaces masked tokens with random tokens in a template, and $P_{thresh}$ is a pre-defined threshold.

To construct the desired template denoted as $T_x$, we follow Algorithm 2. We initiate this process by evaluating the contribution score of each input token through the application of an established interpretable machine learning technique [20] (line 3). Subsequently, we commence with a complete mask template, wherein all tokens are designated for masking (line 4). This initial state fails to satisfy Equation (8), given that the generator would generate entirely random inputs without any discernible token. Next, our iterative procedure involves systematically shifting tokens from a masked to a non-masked state, guided by the contribution scores of each token (lines 9–11). The goal is to achieve a template $T_x$ that conforms to Equation (8). In essence, during the first iteration, we identify the token with the highest contribution score and designate it as non-masked, thereby updating the template accordingly. With this modified template, we generate 1,000 random instances by preserving the current mask configuration. Subsequently, we calculate the probability that these instances yield the same prediction as the original input. If Equation (8) remains unsatisfied, we proceed to the next iteration, marking the token with the second highest contribution score as non-masked. This iterative process continues until Equation (8) is fulfilled. This iterative token selection process is designed to be greedy at each step, consistently opting for the token with the highest contribution score. As a result of this sequential approach, the eventual template $T_x$ that emerges retains the minimal number of tokens from the original input $x$. Moreover, since the input $x$ is an incorrect prediction, the generated template $T_x$ is likely to produce misclassification (i.e., the probability of misclassification is larger than $P_{thresh}$).

*Running Example.* We illustrate the aforementioned process using a practical example. Let's focus on the second seed sentence in Figure 7, which is, "It is always enthralling." To begin, we calculate the contribution score for each token, as depicted in the *Score Visualization* column. Next, our template initially consists of all "[MASK]" tokens. We then evaluate whether this template satisfies Equation (8). If the equation does not hold, we replace the token with the highest contribution score from the "[MASK]" tokens with its original counterpart. In this instance, that token is "enthralling." We reevaluate Equation (8). If it still does not satisfy the equation, we proceed to replace the token with the second-highest contribution score from the "[MASK]" tokens with its original counterpart.

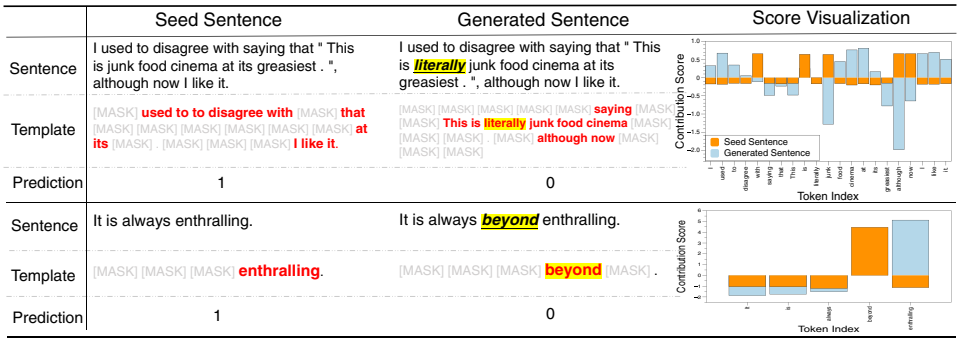


Fig. 7. Visualization of the buggy reason of two ALiCT-generated test cases.

This iterative process continues until Equation (8) is met or until we exhaust all possibilities. In our case, the final template becomes: "[MASK] [MASK] [MASK] [MASK] enthralling." If we substitute the "[MASK]" tokens with random tokens and generate a concrete sentence, this new sentence will possess a probability greater than $P_{thresh}$ of yielding the same prediction as the original sentence.

*Case Study.* We perform a case study to demonstrate the effectiveness of our methodology. Figure 7 shows the two examples of Pass-to-Fail cases. In the *Seed Sentence* column, details about the seed sentence from the dataset are presented (e.g., sentence $x$, identified template $T_x$, and prediction label). The *Generated Sentence* column provides information about the sentence generated by ALiCT. The *Score Visualization* column illustrates the contribution score of each token in the sentence, with blue bars representing the seed sentence and orange bars representing the generated sentence. Modified tokens are emphasized with a yellow background, and identified templates are indicated with red text.

From the results, we have the following observations: (1) The tokens introduced by ALiCT can wield a significant impact, often taking precedence in influencing the model's predictions. This is exemplified in the second case within Figure 7, wherein ALiCT inserts the token *beyond* into the sentence, consequently altering the model's prediction. A thorough examination of the visualization results underscores the significance of the *beyond* token, which commands a substantial contribution score, surpassing even the cumulative effect of other tokens. Furthermore, the validity of this phenomenon is corroborated by the identified template. As stated in Equation (8), the template underscores that sentences adhering to its structure hold a greater than 90% likelihood of eliciting an identical model prediction. This observation reaffirms that the model displays heightened sensitivity toward specific tokens, possibly due to its training dataset's inclination toward these tokens. (2) Another notable observation pertains to instances where the newly introduced token exhibits minimal individual contribution to the score. However, its presence serves to reshape the distribution of contribution scores among other tokens. This phenomenon is exemplified by the first case in Figure 7. Upon the inclusion of the token *Literally*, a notable shift occurs in the contribution scores of the remaining tokens. Furthermore, the preeminent template identification also undergoes significant alteration. Previously characterized by "... used to disagree with...,..., I like it," the dominant template now transforms into "This is literally junk food cinema." Notably, the phrase "I like it" no longer commands substantial influence. This shift subsequently prompts a change in the model's prediction. This observation stems from the intrinsic nonlinearity of machine learning models. Even the most minor perturbation can propagate throughout the system, causing a shift in the impact exerted by other tokens that play a role in the model's prediction. Furthermore, ALiCT has the capability to generate valuable test cases that effectively provoke such changes.

## 7 Threats to Validity

*Internal.* We have identified internal concerns originating from the following three aspects.

First, we implemented generative rules with the intention of amalgamating phrases sourced from the search dataset. The incorporation of specific user-defined phrases into these generative rules may unintentionally result in incomplete coverage of the entire test case distribution for the LC. To address this potential issue, we proactively tackle it by encompassing the full spectrum of test case diversity, leveraging all available phrases from the search dataset. This strategy is based on the assumption that the search dataset accurately mirrors real-world scenarios. By adopting this approach, we aim to minimize the gap between the comprehensive distribution of test cases and those generated by our method, simultaneously enhancing semantic and structural diversity while ensuring alignment between the test cases and the LC.

Second, in order to ensure consistent evaluation, we assigned two participants to label each sentence, with each participant receiving a distinct label. However, this approach introduces the risk of participants mislabeling certain sentences. To mitigate this potential threat, we implemented two measures: first, we randomly selected the sentences assigned to each participant, and second, we tasked the participants with performing each labeling task, aggregating the labels provided by the two participants. Consequently, in accordance with the Law of Large Numbers [14], our results can attain probabilistic correctness when dealing with a large number of randomly selected sentences.

Lastly, the reference corpus and word sentiment utilized in our approach may not be fully representative of all English grammatical structures and word sentiments. To address this potential limitation, we opted for a widely used dataset in the NLP domain [29]. Specifically, we utilized the Penn Treebank [44] dataset for the reference corpus due to its diversity, derived from 98,732 stories from the Wall Street Journal for syntactic annotation. Additionally, we employed the SentiWordNet for the word sentiment dataset, choosing it for its extensive usage in various research projects and licensing to over 300 research groups [3].

*External.* The external threats to validity come from the following aspects: First, ALiCT is both implemented and evaluated based on a specific set of LCs, as outlined in Tables 2 and 3. However, there is a potential risk that this focused evaluation may limit the generalizability of ALiCT. To address this concern, we are undertaking the following measures: (1) We choose a diverse set of LCs for evaluation. These selected capabilities span various applications such as sentiment analysis, hate speech detection, and others (e.g., fairness). We ensure diversity not only in terms of application but also in usage and complexity. (2) The LCs selected for evaluation are not arbitrary; rather, they are well-established and widely used in existing research. This deliberate choice aims to ensure that the evaluation of ALiCT is grounded in linguistic tasks that have proven relevance and applicability in the broader research community.

Second, the evaluation subjects employed in our experiments exclusively consist of English models, potentially limiting the generalizability of ALiCT in multilingual settings. To mitigate this limitation, we are implementing the following strategies: (1) In the design of ALiCT, it is important to note that no English-specific knowledge is mandated. Consequently, in theory, ALiCT possesses the potential for generalization to multilingual settings, as it does not rely on language-specific features during the design phase. (2) Although ALiCT utilizes a BERT-base model for word suggestion in sentence expansion, we note that BERT-base is trained on unlabeled English sentences and may not be optimal for expanding sentences in other languages. However, to enhance multilingual adaptability, the BERT-base model can be substituted with bert-base-multilingual. This alternative model has been trained with data from 104 languages, sourced from the largest Wikipedia, thereby broadening its LCs.

Finally, we have chosen neuron coverage as one of our evaluation metrics to assess the diversity of the generated test inputs. However, certain existing studies have cast doubts on the efficacy of neuron coverage as an objective function for generating adversarial examples [22, 74, 75]. It is crucial to note that these studies do not outright dismiss the effectiveness of neuron coverage as a metric for measuring diversity. For instance, [74] found that indiscriminately increasing neuron coverage can have a detrimental effect, resulting in the production of less natural inputs and introducing bias in output distribution. In our evaluation, we consciously avoid using coverage as the primary objective in our approach to generating test inputs. Consequently, the concern that test inputs generated by our tool may be less natural does not apply. Furthermore, [75] observed that neuron coverage may not be effective in adversarial settings. It is crucial to highlight that our diversity evaluation is not conducted in adversarial settings; we do not iteratively query the model until errors are found. Thus, our choice of neuron coverage could still represent the diversity of the generated test suite to some degree.

## 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 and Applications.* **Deep Neural Networks (DNNs)** have significantly improved various NLP applications, including reading comprehension, hate speech detection, and machine translation. For instance, word embeddings [30, 46, 52] distributes the semantic of words into numeric vectors, which are then utilized to train neural networks for classification tasks. Meanwhile, Seq2Seq [19, 63, 67] presents an encoder–decoder neural network architecture that has been widely adopted for modeling the sequence generation task, particularly in machine translation and question answering applications. In addition, Google [72] has introduced the attention mechanism, namely transformer, can greatly enhance the accuracy of the generated texts. Accordingly, self-supervised learning paradigm has been applied to the transformer, and it is used for pre-training language model before being fine-tuned or used for specific downstream tasks [13, 54]. Pre-training becomes a crucial step in creating powerful and effective NLP models.

In recent times, it has been observed that scaling pre-trained language models can significantly enhance the model's performance on downstream tasks. As a result, numerous LLMs have been introduced, and these models have exhibited remarkable abilities in solving a wide array of complex tasks. [11, 55, 69]

*Machine Learning Testing and 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 [34, 35]. Therefore, researchers have developed a series of techniques to test ML-based applications. For example, DeepExplore [51] utilizes neuron coverage to partition the input space. It assumes that inputs that share similar neuron coverage belong to the same class. Ma et al. assess the neural coverage of activated neurons in a DNN by drawing an analogy to code branches in traditional software testing [41, 73]. Tian et al. [68] finds erroneous behavior of DNN by generating test inputs that maxmize the neural coverage of activated neurons in the domain of autonomous driving. DeepMutation [42] proposes the mutation testing framework for DNNs. It introduces a set of fault injection operators to perturbate the decision logic of a DNN. DeepStellar [15] relies on state modeling and presents a series of metrics for RNNs. These metrics are used for testing and detecting adversarial examples. AsFault [18] evaluates self-driving car software by automatically generating virtual scenario and searching their parameters toward safety-critical scenarios. Kim et al. [31] measures the difference in deep learning system's behavior between an input and the training data to measure the surprise of the input based on the training data. CRADLE [53] concentrates on

the localization of bugs in deep learning software libraries. In addition, Simin et al. [8, 10] enables energy efficient performance testing for DNNs such with respect to latency degradation and energy consumption degradation.

In recent years, researchers have investigated the occurrence of bugs produced by neural networks in NLP applications, inspired by the work on adversarial examples in computer vision. TestBugger [36] proposes a gradient-guided approach to generate test inputs for identifying bugs in NLP models used for classification tasks. Rel et al. [56] generates adversarial input text by replacing input words with synonyms searching from word saliency and classification probability. Zang et al. [77] introduces word-level adversarial attack model for text classification by sememe-based word substitution and a specific searching algorithm. Li et al. [38] utilizes BERT to identify semantic-preserving word substitutes for adversarial attacking words in the input text. Ebrahibi et al. [16, 17] provides input text transformation operations for character-level NLP models. Zou et al. [82] generates adversarial examples to attack neural machine translation model using reinforcement learning. In addition to evaluating the robustness of NLP applications through NLP model attacks, various other perspectives of these applications are also assessed for their practical utility. Neural machine translation models are evaluated by generating adversarial examples [79, 82] and measuring metamorphic relations between input and translation results [21, 23, 24, 66]. Chen et al. [9] focuses on generating test inputs that can expose energy efficiency degradation of neural machine translation. In addition, Ma et al. [43] assess fairness violations by perturbing human-related noun words and measuring the discrepancy in the model's outputs between the perturbed texts. Our approach differs from existing work in that we concentrate on testing the LCs of NLP applications in an automatic manner, a topic that has yet to be explored.

## 9 Conclusions

This article introduces ALiCT, a tool designed to automate the process of generating test cases for NLP models. Through the utilization of LC specification-driven structural predicates and generative rules, it can automatically create seed test cases. ALiCT also employs syntax-based expansion to further broaden the array of syntactic structures originating from the seed test cases. This ensures a strong alignment between the generated test cases and their LCs, labels, and semantics and enhances the diversity of the seed test cases.

We assess the efficacy of ALiCT across two prominent NLP tasks. Our experiments show that, when measured using Self-BLEU and SD, the test cases generated by ALiCT exhibit a diversity increase of at least 190% in semantic and 2213% more diverse in syntactic aspects compared to those generated by state-of-the-art techniques. This substantial diversity improvement suggests that ALiCT's test cases enhance neuron coverage and introduce a greater number of model failures in 22 out of 25 LCs over the two NLP tasks. Furthermore, we performed a study to validate that ALiCT consistently generates test cases with accurately aligned labels, corresponding LCs, and the semantic context of the expanded test cases. We conducted a thorough analysis of cases that induce failures, uncovering the underlying causes of these issues. Additionally, we demonstrated that ALiCT is applicable for evaluating LLM over LCs. This validates the correctness and practical value of ALiCT in facilitating model evaluation.

Looking ahead, there is a need for additional research stemming from this study, particularly in the domain of LC specification analysis. First, we anticipate that assessing LC through an NLP task could pinpoint specific aspects of erroneous behavior of NLP models, ultimately aiding in their debugging. Additionally, the automation of LC specification generation could significantly facilitate the generation of seed test cases based on natural language descriptions.

## References


[1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating Natural Language Adversarial Examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 2890–2896. DOI: https://doi.org/10.18653/v1/D18-1316

[2] Muhammad Hilmi Asyrofi, Zhou Yang, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, and David Lo. 2022. BiasFinder: Metamorphic Test Generation to Uncover Bias for Sentiment Analysis Systems. *IEEE Transactions on Software Engineering* 48, 12 (2022), 5087–5101. DOI: https://doi.org/10.1109/TSE.2021.3136169

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2010. 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)*. European Language Resources Association (ELRA), Valletta, Malta. Retrieved from http://www.lrec-conf.org/proceedings/lrec2010/pdf/769_Paper.pdf

[4] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. 2021. Cats are not Fish: Deep Learning Testing Calls for Out-of-Distribution Awareness. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE'20)*. ACM, New York, NY, 1041–1052. DOI: https://doi.org/10.1145/3324884.3416609

[5] Som S. Biswas. 2023. Potential Use of Chat GPT in Global Warming. *Annals of Biomedical Engineering* 51, 6 (2023), 1126–1127.

[6] Calculator.net. 2023. Sample Size Calculator. Retrieved from https://www.calculator.net/sample-size-calculator.html

[7] Simin Chen, Soroush Bateni, Sampath Grandhi, Xiaodi Li, Cong Liu, and Wei Yang. 2020. *DENAS: Automated Rule Generation by Knowledge Extraction from Neural Networks*. ACM, New York, NY, 813–825. DOI: https://doi.org/10.1145/3368089.3409733

[8] Simin Chen, Mirazul Haque, Cong Liu, and Wei Yang. 2022a. DeepPerform: An Efficient Approach for Performance Testing of Resource-Constrained Neural Networks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[9] Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. 2022b. 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*. 1148–1160.

[10] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. 2022c. NICGSlowDown: Evaluating the Efficiency Robustness of Neural Image Caption Generation Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15365–15374.

[11] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research* 24, 240:1–240:113. Retrieved from http://jmlr.org/papers/v24/22-1144.html

[12] Xavier Suau Cuadros, Luca Zappella, and Nicholas Apostoloff. 2022. Self-Conditioning Pre-Trained Language Models. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 162. Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 4455–4473. DOI: https://proceedings.mlr.press/v162/cuadros22a.html

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019a. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'19)*, Vol. 1, Long and Short Papers. Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. DOI: https://doi.org/10.18653/V1/N19-1423

[14] W. J. Dixon and Frank J. Massey. 1951. *Introduction to Statistical Analysis*. McGraw-Hill, New York, NY, 370.

[15] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. Deepstellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 477–487.

[16] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 2018a. On Adversarial Examples for Character-Level Neural Machine Translation. In *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, Santa Fe, New Mexico, USA, 653–663. Retrieved from https://aclanthology.org/C18-1055

[17] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018b. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Vol. 2 (Short Papers). Association for Computational Linguistics, Melbourne, Australia, 31–36. DOI: https://doi.org/10.18653/v1/P18-2006

[18] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically Testing Self-Driving Cars with Search-Based Procedural Content Generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 318–328.

[19] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1243–1252.

[20] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. Lemna: Explaining Deep Learning Based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 364–379.

[21] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. 2020. Machine Translation Testing via Pathological Invariance. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 863–875.

[22] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks? In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Retrieved from https://api.semanticscholar.org/CorpusID:210146632

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

[24] Pinjia He, Clara Meister, and Zhendong Su. 2021. Testing Machine Translation via Referential Transparency. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE'21)*. IEEE, 410–422.

[25] Chaitra V. Hegde and Shrikumar Patil. 2020. Unsupervised Paraphrase Generation Using Pre-Trained Language Models. arXiv:2006.05477. Retrieved from https://arxiv.org/abs/2006.05477

[26] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing.

[27] HuggingFace. 2022. HuggingFace. Retrieved from https://huggingface.co

[28] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepCrime: Mutation Testing of Deep Learning Systems Based on Real Faults. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'21)*. Cristian Cadar and Xiangyu Zhang (Eds.). ACM, New York, NY, 67–78. DOI: https://doi.org/10.1145/3460319.3464825

[29] Mujtaba Husnain, Malik Muhammad Saad Missen, Nadeem Akhtar, Mickaël Coustaty, Shahzad Mumtaz, and VB Prasath. 2021. A Systematic Study on the Role of SentiWordNet in Opinion Mining. *Frontiers of Computer Science* 15, 4 (2021), 1–19.

[30] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL'17)*, Vol. 2, Short Papers. Mirella Lapata, Phil Blunsom, and Alexander Koller (Eds.). Association for Computational Linguistics, 427–431. DOI: https://doi.org/10.18653/V1/E17-2068

[31] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE'19)*. IEEE, 1039–1049.

[32] Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual Constituency Parsing with Self-Attention and Pre-Training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 3499–3505. DOI: https://doi.org/10.18653/v1/P19-1340

[33] Nikita Kitaev and Dan Klein. 2018. Constituency Parsing with a Self-Attentive Encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Vol. 1, Long Papers. Association for Computational Linguistics, Melbourne, Australia, 2676–2686. DOI: https://doi.org/10.18653/v1/P18-1249

[34] Fred Lambert. 2016. Understanding the Fatal Tesla Accident on Autopilot and the NHTSA Probe. *Electrek*, July 1.

[35] Sam Levin. 2018. Tesla Fatal Crash: 'Autopilot' Mode sped up car Before Driver Killed, Report Finds. *The Guardian*, June 8.

[36] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS'19)*. The Internet Society. Retrieved from https://www.ndss-symposium.org/ndss-paper/textbugger-generating-adversarial-text-against-real-world-applications/

[37] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020a. BERT-ATTACK: Adversarial Attack Against BERT Using BERT. arXiv:2004.09984. Retrieved from https://arxiv.org/abs/2004.09984

[38] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020b. BERT-ATTACK: Adversarial Attack Against BERT Using BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6193–6202. DOI: https://doi.org/10.18653/v1/2020.emnlp-main.500

[39] Alexander Ligthart, Cagatay Catal, and Bedir Tekinerdogan. 2021. Systematic Reviews in Sentiment Analysis: A Tertiary Study. *Artificial Intelligence Review* 54, 4997–5053. Retrieved from https://api.semanticscholar.org/CorpusID:233769825

[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692. Retrieved from http://arxiv.org/abs/1907.11692

[41] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao and Yadong Wang. 2018a. Deepgauge: Multi-Granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.

[42] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018b. Deepmutation: Mutation Testing of Deep Learning Systems. In *Proceedings of the IEEE 29th international symposium on software reliability engineering (ISSRE'18)*. IEEE, 100–111.

[43] Pingchuan Ma, Shuai Wang, and Jin Liu. 2020. Metamorphic Testing and Certified Mitigation of Fairness Violations in NLP Models. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI'20)*. Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 458–465. DOI: https://doi.org/10.24963/ijcai.2020/64

[44] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19, 2 (Jun. 1993), 313–330.

[45] Binny Mathew, Punyajoy Saha, Seid Muhie Yimam, Chris Biemann, Pawan Goyal, and Animesh Mukherjee. 2021. HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 14867–14875.

[46] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the 1st International Conference on Learning Representations (ICLR'13)*, Workshop Track Proceedings. Yoshua Bengio and Yann LeCun (Eds.). Retrieved from http://arxiv.org/abs/1301.3781

[47] John X. Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks in Natural Language Processing. arXiv:2005.05909. Retrieved from https://arxiv.org/abs/2005.05909

[48] OpenAI. 2023a. GPT-4 Technical Report. arXiv:2303.08774. DOI: https://doi.org/10.48550/ARXIV.2303.08774

[49] OpenAI. 2023b. GPT Model Documentation. Retrieved from https://platform.openai.com/docs/introduction

[50] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Philadelphia, PA, USA, 311–318. DOI: https://doi.org/10.3115/1073083.1073135

[51] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, New York, NY. DOI: https://doi.org/10.1145/3132747.3132785

[52] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543.

[53] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. 2019. CRADLE: Cross-Backend Validation to Detect and Localize Bugs in Deep Learning Libraries. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE'19)*. IEEE, 1027–1038.

[54] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. Retrieved from https://api.semanticscholar.org/CorpusID:49313245

[55] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI Blog* 1, 8 (2019), 9. Retrieved from https://api.semanticscholar.org/CorpusID:160025533

[56] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 1085–1097. DOI: https://doi.org/10.18653/v1/P19-1103

[57] Marco Tulio Ribeiro. 2023. CHECKLIST Github Repository. Retrieved from https://github.com/marcotcr/checklist/tree/master

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

[59] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Association for Computational Linguistics (ACL)*.

[60] Paul Röttger, Haitham Seelawi, Debora Nozza, Zeerak Talat, and Bertie Vidgen. 2022. Multilingual HateCheck: Functional Tests for Multilingual Hate Speech Detection Models. In *Proceedings of the 6th Workshop on Online Abuse and Harms (WOAH)*. Kanika Narang, Aida Mostafazadeh Davani, Lambert Mathias, Bertie Vidgen, and Zeerak Talat (Eds.). Association for Computational Linguistics, Seattle, WA (Hybrid), USA, 154–169. DOI: https://doi.org/10.18653/v1/2022.woah-1.15

[61] Paul Röttger, Bertie Vidgen, Dong Nguyen, Zeerak Waseem, Helen Margetts, and Janet Pierrehumbert. 2021. HateCheck: Functional Tests for Hate Speech Detection Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Vol. 1, Long Papers. Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 41–58. DOI: https://doi.org/10.18653/v1/2021.acl-long.4

[62] Anna Schmidt and Michael Wiegand. 2017. A Survey on Hate Speech Detection Using Natural Language Processing. In *Proceedings of the 5h International Workshop on Natural Language Processing for Social Media*. Association for Computational Linguistics, Valencia, Spain, 1–10. DOI: https://doi.org/10.18653/v1/W17-1101

[63] Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[64] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, WA, USA, 1631–1642. DOI: https://aclanthology.org/D13-1170

[65] Ezekiel O. Soremekun, Sakshi Udeshi, and Sudipta Chattopadhyay. 2022. Astraea: Grammar-Based Fairness Testing. *IEEE Transactions on Software Engineering* 48, 12 (2022), 5188–5211. DOI: https://doi.org/10.1109/TSE.2022.3141758

[66] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic Testing and Improvement of Machine Translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 974–985.

[67] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems* 27, 3104–3112.

[68] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering*. 303–314.

[69] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Roziére, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971. DOI: https://doi.org/10.48550/ARXIV.2302.13971

[70] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated Directed Fairness Testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE'18)*. Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, New York, NY, 98–108. DOI: https://doi.org/10.1145/3238147.3238165

[71] Sakshi Udeshi and Sudipta Chattopadhyay. 2021. Grammar Based Directed Testing of Machine Learning Systems. *IEEE Transactions on Software Engineering* 47, 11 (2021), 2487–2503. DOI: https://doi.org/10.1109/TSE.2019.2953066

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. *Advances in Neural Information Processing Systems* 30, 5998–6008.

[73] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. Deephunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 146–157.

[74] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. [n. d.]. Correlations Between Deep Neural Network Model Coverage Criteria and Model Quality. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20)*. DOI: https://doi.org/10.1145/3368089.3409671

[75] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2022. Revisiting Neuron Coverage Metrics and Quality of Deep Neural Networks. In *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'22)*. IEEE, 408–419. DOI: https://doi.org/10.1109/SANER53432.2022.00056

[76] Ping Yu, Yang Zhao, Chunyuan Li, and Changyou Chen. 2021. Rethinking Sentiment Style Transfer. In *Proceedings of the Findings of the Association for Computational Linguistics: EMNLP'21*, Marie-Francine Moens, Xuanjing Huang,

Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Punta Cana, Dominican Republic, 1569–1582. DOI: https://doi.org/10.18653/v1/2021.findings-emnlp.135

[77] Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020a. Word-Level Textual Adversarial Attacking as Combinatorial Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 6066–6080. DOI: https://doi.org/10.18653/v1/2020.acl-main.540

[78] Ruiyi Zhang, Changyou Chen, Zhe Gan, Zheng Wen, Wenlin Wang, and Lawrence Carin. 2020. Nested-Wasserstein Self-Imitation Learning for Sequence Generation. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS'20)*, Proceedings of Machine Learning Research, Vol. 108. Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 422–433. Retrieved from http://proceedings.mlr.press/v108/zhang20b.html

[79] Xinze Zhang, Junzhe Zhang, Zhenhua Chen, and Kun He. 2021. 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,* Vol. 1, Long Papers. Association for Computational Linguistics, Online, 1967–1977. DOI: https://doi.org/10.18653/v1/2021.acl-long.153

[80] Binggui Zhou, Guanghua Yang, Zheng Shi, and Shaodan Ma. 2022. Natural Language Processing for Smart Healthcare. *IEEE Reviews in Biomedical Engineering* 17, 4–18.

[81] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A Benchmarking Platform for Text Generation Models. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'18)*. ACM, New York, NY, 1097–1100. DOI: https://doi.org/10.1145/3209978.3210080

[82] Wei Zou, Shujian Huang, Jun Xie, Xinyu Dai, and Jiajun Chen. 2020. A Reinforced Generation of Adversarial Examples for Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 3486–3497.

Received 7 September 2023; revised 6 February 2024; accepted 9 May 2024