TestAug: A Framework for Augmenting Capability-based NLP Tests

Guanqun Yang▲ Mirazul Haque♢ Qiaochu Song▲ Wei Yang♢ Xueqing Liu▲
▲Department of Computer Science, Stevens Institute of Technology
♢Department of Computer Science, The University of Texas at Dallas
gyang16@stevens.edu, mirazul.haque@utdallas.edu, qsong6@stevens.com
wei.yang@utdallas.edu, xliu127@stevens.edu

Abstract

The recently proposed capability-based NLP testing allows model developers to test the functional capabilities of NLP models, revealing functional failures that cannot be detected by the traditional heldout mechanism. However, existing work on capability-based testing requires extensive manual efforts and domain expertise in creating the test cases. In this paper, we investigate a low-cost approach for the test case generation by leveraging the GPT-3 engine. We further propose to use a classifier to remove the invalid outputs from GPT-3 and expand the outputs into templates to generate more test cases. Our experiments show that TestAug has three advantages over the existing work on behavioral testing: (1) TestAug can find more bugs than existing work; (2) The test cases in TestAug are more diverse; and (3) TestAug largely saves the manual efforts in creating the test suites. The code and data for TestAug can be found at https://github.com/guanqun-yang/testaug.

1 Introduction

In recent years, natural language processing (NLP) has seen major breakthroughs in the model performances. Conventional approaches to evaluating NLP models’ performance rely on reporting aggregate metrics such as the accuracy and F-1 scores on the held-out dataset. However, the held-out scores do not represent the model’s performance on data in the wild (Geva et al., 2019; Gururangan et al., 2018; Bai et al., 2021). Moreover, the aggregated scores cannot shed lights on where the model fails and how to fix the failures. For example, recent studies show that even stress-tested commercial NLP APIs (e.g., Google Cloud’s Natural Language API† and Microsoft’s Text Analytics API‡) often fail on test cases of simple behaviors (Glockner et al., 2018; Ribeiro et al., 2020).

To assist developers in finding behavioral failures in their models, recent work has proposed a framework called CheckList for behavioral or capability-based NLP testing (Ribeiro et al., 2020; Tarunesh et al., 2021). Such tests include input and output pairs to examine the model’s performance on each linguistic capability. Table 1 shows examples of capability-based tests for three NLP tasks. For example, the test case “No one loves the food” contains a negated positive word, and its sentiment is negative. With a set of sentences each containing negated positive words, we can test whether the model correctly understands the sentiment of any sentence containing the negated positive word. Such a set is called a test suite.

In existing capability-based testing frameworks (Ribeiro et al., 2020; Tarunesh et al., 2021), the test suites are generated from manually created natural language templates (e.g., “She is a [] person” and ”she is not a [] person.”) and a pre-defined list of words to fill the template (e.g., generous, mean). Existing work thus has two disadvantages:

- High Cost of Labeling. The current practice of creating templates requires a high cost. Even worse, despite the crowdsourcing avail-

Table 1: Example of capability-base tests on three NLP tasks: sentiment classification, paraphrase detection, and natural language inference.

<table>
<thead>
<tr>
<th>Task</th>
<th>Capability</th>
<th>Test Description</th>
<th>Input</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentiment Classification</td>
<td>Negation</td>
<td>Positive if negated</td>
<td>No one loves the food.</td>
<td>Negative</td>
</tr>
<tr>
<td>Paraphrase Detection</td>
<td>Negation</td>
<td>Paraphrase if replacing a word with the negated antonym</td>
<td>She is a generous person. She is not a mean person.</td>
<td>Paraphrase</td>
</tr>
<tr>
<td>Natural Language Inference</td>
<td>Downward entailment</td>
<td>Entailment if replacing a word with its superset</td>
<td>Some cows are brown. Some animals are brown.</td>
<td>Entailment</td>
</tr>
</tbody>
</table>
ability, expert annotations are often required: the templates need to both follow the linguistic rules and capture potential NLP pitfalls.

- **Low Diversity.** Even when expert annotations are available, the test cases generated following the current practice often only show diversity on a superficial level.

  For example, for some capabilities in (Ribeiro et al., 2020), the only variation comes from persons’ names (e.g., "If {male name} and {female name} were alone, do you think he would reject her?" and "If {male name} and {female name} were alone, do you think she would reject him?"). This lack of diversity hinders the test cases from revealing more of the models’ prediction errors when they satisfy the test.

In this work, we propose a novel framework for capability-based testing to address the challenges of scalability and diversity mentioned above. In our framework, TestAug, the developer first annotates a few seed test cases, TestAug then leverages the GPT-3 engine (Brown et al., 2020) to generate test cases similar to the seed. Next, TestAug expands the GPT-3 generated cases into templates to generate more cases. Finally, TestAug includes a validity classifier to check the correctness of the generated cases, and discard the invalid cases. Our experiments show that the validity classifiers filter the invalid cases with success rates of at least 90%, 90%, and 80% for three tasks we evaluated. Furthermore, the valid generated cases are more diverse and can detect more bugs than existing work (Ribeiro et al., 2020). Our contributions are three folds:

- We propose a novel framework TestAug for automatically generating capability-based NLP test suites based on GPT-3;

- TestAug is shown to outperform the existing capability-based NLP testing framework in 3 aspects: better ability to detect bugs, more diversity, and fewer annotation efforts;

- We have published our test suite to help developers and researchers test their NLP models;

## 2 Background

**Capability-based Testing for NLP Models.** Traditionally, NLP models are evaluated using the held-out datasets, that is, using the train/validation/test split. However, recent studies (Yanaka et al., 2019; Bowman and Dahl, 2021) found that the held-out mechanism suffers from bias (Poliak et al., 2018) and cannot effectively reflect the improvements in the model performance (Yanaka et al., 2019). To help gain a more comprehensive understanding of the model performance, researchers proposed a new approach to evaluating NLP models called linguistic capability-based testing (Ribeiro et al., 2020; Joshi et al., 2020a; Tarunesh et al., 2021). That is, instead of testing and reporting the average performance on one dataset, we test and report multiple metrics by assessing the model’s capabilities of handling different test scenarios. The taxonomy of the capabilities can be organized by linguistic theory (?), logic, domain knowledge (Joshi et al., 2020b), or the functional requirements defined by the specific application (Kirk et al., 2021; Wang et al., 2021; van Aken et al., 2021). For example, to test an NLI model’s logic reasoning capabilities, researchers examined its different aspects, such as handling of negations, boolean, quantifiers, comparatives, monotonicity, etc. (Richardson et al., 2020; ?). Later, Ribeiro et al. (2020) extended capability-based testing to other NLP tasks, including sentiment classification, paraphrase detection, and question answering. The capabilities for testing would be listed by software developers or by the subject matter experts who manually identify a taxonomy of errors based on their expertise in data annotation (Röttger et al., 2021). The construction method for the test suites can be divided into fully manual (?Joshi et al., 2020a) and semi automatic approaches. The manual approaches often suffer from scalability issues (?). Some existing approaches proposed to scale up the annotation by leveraging non-expert annotators, but had to restrict the capabilities to avoid making the tasks too complicated for the annotators (Joshi et al., 2020a). To construct a massive scale test suite without large manual annotation efforts, Poliak et al. Poliak et al. (2018) proposed to recast 13 existing datasets on 7 different tasks (e.g., NER, relation extraction) into a unified NLI test suite, but this approach does not apply to other NLP tasks. Other works remedy the scalability issue by manually coming up with templates where the blanks can be filled with interchangeable tokens or a cloze-style prediction from language models (Ribeiro et al., 2020; Tarunesh et al., 2021), but automatically generating the tem-
plates remain a challenging task (Tarunesh et al., 2021; Jeretic et al., 2020). Finally, Salvatore et al. (2019) proposed a formal language for generating templates, although it can be used to generate examples of contradictions in NLI. In contrast to the previous work, we propose to leverage the generative power of GPT-3 to fully automate the construction of capability-based test suites. Our framework thus overcomes the scalability issue in existing work.

**Prompt Learning for GPT-3.** Our work has employed the GPT-3 engine (Brown et al., 2020) for the generation and verification of the test suites, where we have manually engineered and optimized the prompt messages (Section 4). Prompt learning was found to be helpful for a wide range of tasks (Shin et al., 2020; Gao et al., 2021), including major natural language generation tasks (Li and Liang, 2021). To the best of our knowledge, however, there only exist a few works in literature that systematically investigated prompt learning for GPT-3 generation. Mishra et al. (2021) proposed a dataset for teaching GPT-3 and BART (Lewis et al., 2020) to follow instructions. Reynolds and McDonell (2021a) summarized the essential findings in prompt engineering for GPT-3 from blogs and social media and found that few-shot demonstration can be worse than zero-shot demonstration for GPT-3. Due to the scarcity of literature, we propose a new framework for prompting GPT-3 to generate the capability-based test suites (Section 4).

### 3 Problem Definition

Software testing refers to the process of identifying the inconsistencies between software’s actual and expected execution process (Zhang et al., 2019). Software testing includes white-box testing and black-box testing. The latter is also known as behavioral testing, which examines the external behaviors of the software. It often requires the developers to collect test cases (i.e., input/output pairs) to constitute a test suite (i.e., a collection of cases for testing specific software behaviors).

In recent years, following the success of natural language processing, behavioral testing was introduced to test NLP models (Ribeiro et al., 2020), especially large language models that show state-of-the-art performance. The expected behaviors of NLP models were defined in several aspects, which are called the capabilities of the models. For example, for a sentiment classification model, we should expect it to output the negative sentiment for an input sentence containing a negated positive word, e.g., *I don’t like the food*. Behavioral testing goes beyond the held-out validation evaluation scheme, allowing software developers to detect and monitor the behavioral failures of the model on top of the performance metrics on the held-out dataset, providing insights into the model behavior in multiple aspects.

The most recent work on NLP behavioral testing is called CheckList (Ribeiro et al., 2020). In CheckList, around 10 capabilities are defined for each NLP task being tested. For each capability, several tests were created by the developer, and the requirement of each test is described with a natural language description as in Table 1. Each test contains one or more natural language templates containing slots, with a pre-defined word list associated with each slot. For example, for the aforementioned negation capability, one test template is "*[it] [benot] [a:pos_adj] [air_noun]*." By defining a list for each slot, the developer can use this template to generate test cases such as "**That is not a perfect seat.**" The test cases generated following each test and the overarching linguistic capability constitute the test suite $T$.

### 4 The TestAug Framework

Given a linguistic capability and their specific tests, the previous approach to generating test cases relies on manual templates. In this paper, we propose a novel framework (namely, TestAug for Test Suite Augmentation) to reduce such manual efforts. Figure 1 shows the control flow of our framework. First, TestAug starts with a few seed test cases from the CheckList test suite (Ribeiro et al., 2020). It leverages the description of the test and the seed test cases to prompt GPT-3 to generate more cases (Section 4.1). The correctness of the generated GPT-3 cases is examined through a trained binary classifier (Section 4.2), and expanded into more templates by matching the GPT-3 case with the seed case (Section 4.3). Finally, the aggregate test suite is used for model testing; the test results provide feedback to the NLP model developer for the next iteration of testing.

#### 4.1 Prompt Engineering for Instructing GPT-3 to Generate Test Cases

We design our natural language inputs (i.e., prompts) based on the practices of instructing GPT-3 for dataset creation (Liu et al., 2022; Reif et al.,
We use prompt engineering (Liu et al., 2021) to instruct GPT-3 \(^3\) to generate test cases that meet the requirement of the test description. A prompt is an instructive sentence that tells GPT-3 the command to follow. For example, Table 2 shows the prompt "A negative sentiment sentence with negated positive word." Meanwhile, it has been shown that generative models’ performance can be improved with demonstrations (Gao et al., 2021), i.e., example sentences that append the prompt sentence to show examples of what the generated sentence should look like. For example, for the prompt mentioned above, one demonstration sentence is "No one enjoys that seat". In this paper, to instruct GPT-3 to generate test cases that meet the requirement of the test description, we propose to simply use the test description as the prompt, followed by three randomly sampled seed test cases from the CheckList test suite (Ribeiro et al., 2020; Tarunesh et al., 2021) as the demonstrations. We use the dashed points to format each demonstrated case, wrapped by the bracket "{}". Our prompt design for the sentiment classification task can be found in Table 2; the one for paraphrase detection and natural language inference can be found in Table 7 in the Appendix. In particular, our choice for the format (especially the bracket) comes from our empirical observation that such a format encourages GPT-3 to generate valid sentences with higher probabilities. If skipping the bracket, GPT-3 tends to generate long sentences that are dissimilar to the demonstrations; for the two-sentence tasks, GPT-3 is less likely to generate correctly formatted pairs without the bracket.

4.2 Filtering Incorrect Test Cases

The test cases generated by GPT-3 may fail to satisfy tests as they (1) do not satisfy the required format; for example, the tasks of paraphrase detection and natural language inference require a pair of sentences as a test case while sometimes only one sentence could be found in the GPT-3 generation, (2) do not satisfy the tests expressed in the prompts (i.e., the requirement or description of the test case); for example, the generated test case ("Joe isn’t at the party.", "Joe is at the party.") for natural language inference is incorrect as it violates the required label "entailment" for natural language inference task; the "This food isn’t bad, but I wasn’t expecting much." for sentiment classification does not meet the requirement for the test description "I thought something was negative, but it was neutral." because the former part is not negative, neither is the latter neutral.

To address the issues above, we create a validity filter that automatically removes the invalid test cases generated by GPT-3. The filter is constructed by training a binary classifier. The training data for the binary classifier comes from our manual annotation of the validity of the GPT-3 generated sentences. We follow the following two-phase process for the filter. In the first phase, we instruct GPT-3 to generate 30-50 cases for each test description, and two annotators manually annotated the validity of the test case by checking the two types of errors discussed in Section 4.2\(^4\). The tests

---

\(^3\)More specifically, we use a GPT-3 variant (namely, davinci-instruct-beta) that specializes in following instructions for better its generation quality in our pilot experiments.

\(^4\)The annotators were two of the authors; they are both graduate students in computer science working on NLP-
whose test cases were predominately valid\(^5\) were expected to generate valid cases most of the time; otherwise, we proceeded to the second phase. In the second phase, we keep instructing GPT-3 to generate sentences until at least 100 invalid and 100 valid cases were collected. The sentences in the second phase were then used as the training set to fine-tune a \texttt{roberta-base} classifier, while the sentences in the first phase were used as the test set. Afterward, we could use the trained classifiers together with GPT-3 to generate test cases in a fully automatic manner.

4.3 Expanding GPT-3 Generated Test Cases into Templates

After obtaining the generated test cases from GPT-3, we can further augment the test suite by expanding the GPT-3 generated cases into templates. More specifically, if a word in a seed test case reappears in the GPT-3 generated test cases, it can be converted back into the slot, and we can vary the slot words using the pre-defined list. For example, using the template "No one [pos_verb_present]s [the] [air_noun]." from CheckList and the pre-defined word "appreciates" for [pos_verb], we create the seed sentence "No one [appreciates] that airline.". By demonstrating this seed sentence to GPT-3, it generates the new sentence "No one [appreciates] that air traffic controller.". Since "appreciate" appears again, we can convert it back to the slot [pos_verb], resulting in a new template "No one [pos_verb] that air traffic controller.". As misplaced pronouns yield nonsensical sentences, we only take the nouns, verbs, and adjectives (i.e., content words) into account when creating new templates; for example, even though "that" also reappears in the generated sentence, we do not create a new slot at its location.

5 Experiments

In this section, we evaluate the effectiveness of TestAug and compare it with existing work (Ribeiro et al., 2020; Tarunesh et al., 2021) in multiple aspects. First, we compare TestAug’s ability to detect the model failures with existing work (Section 5.2). Second, we quantitatively investigate the diversity of test cases (Section 5.3). Since test cases are automatically generated with TestAug, we also investigate the validity of the generated cases, e.g., whether the generated test cases correctly satisfy each capability (Section 5.4). Finally, we quantitatively evaluate the manual efforts saved by TestAug compared to CheckList (Section 5.5). Before reporting these results, we first explain our experimental settings in Section 5.1.

5.1 Experiment Settings

Evaluated Tasks. We compare our framework with existing works by following their experiment settings (Ribeiro et al., 2020; Tarunesh et al., 2021). We investigate the following NLP tasks: sentiment classification (i.e., the Stanford Sentiment Treebank (SST) dataset\(^6\) (Socher et al., 2013)), paraphrase detection (i.e., the Quora Question Pair (QQP) dataset\(^7\)), and natural language inference (i.e., the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015)). Existing work also studied extractive question answering (Ribeiro et al., 2020) and hate speech detection (Röttger et al., 2021); in this paper, however, we skip the two tasks for the following reasons. We skip extractive QA because we find it empirically challenging for GPT-3 to generate examples where all the required components (context, question, answer) meet the requirements at the same time. We thus leave extractive QA for future work. We skip hate speech detection because GPT-3 cannot be prompted for generating profanity words\(^8\).

Evaluated Models. Existing work (Ribeiro et al., 2020) evaluated their capability-based testing frameworks on state-of-the-art NLP models such as BERT, RoBERTa, and commercial APIs such as Google Cloud’s Natural Language API or Microsoft’s Text Analytics API. In this paper, we focus on testing only the open-source models because the underlying model of these APIs are inaccessible for us to fine-tune, which is critical in our evaluation. For all our three tasks (i.e., SST, QQP, and SNLI), there exist publicly available fine-tuned models on the HuggingFace model Hub\(^9\); thus, we reuse these fine-tuned models to test our framework. A complete list of models we evaluate can

---

\(^5\)We used discretized binary version – SST2.

\(^6\)https://www.kaggle.com/c/quora-question-pairs

\(^7\)https://huggingface.co/models

\(^8\)Our attempts to generate profanity words were denied with a flagged warning message from GPT-3: “These statements are all incredibly harmful and oppressive. They promote hatred and bigotry against a marginalized group of people, and they should not be tolerated.”

\(^9\)https://huggingface.co/models
be found in Table 10.

5.2 Evaluating TestAug’s Ability for Bug Detection

5.2.1 Metric

Patch Failure Rate. To the best of our knowledge, we are not aware of any existing method that directly compares the effectiveness of two NLP test suites. One may think the most straightforward approach is directly comparing the failure rates of the same model on the two test suites. Despite the simplicity, we argue that these two failure rates are in fact incomparable: the effectiveness of a test suite is defined by how many bugs it can find (Kochhar et al., 2015); as a result, it is unclear whether a test suite with a lower failure rate but more error cases has a better performance.

To compare the effectiveness of test suites $T_1, T_2, \ldots, T_N$, we propose to create new training and testing sets as follows.

$$T_{\text{Test}} \leftarrow \text{Sample}(\bigcup_{n=1}^{N} T_n), \quad T_{\text{Train}}^{(n)} \leftarrow T_n - T_{\text{Test}}$$

Then we use the training sets $T_{\text{Train}}^{(n)}$ to patch (or fine-tune) the model $M$ and evaluate the patched (or fine-tuned) model $\hat{M}_n$ on the test set $T_{\text{Test}}$. We compare a model $\hat{M}_n$’s patched failure rate on different training sets; a lower rate thus indicates a stronger capability in finding bugs.

$$\text{FR}_{\text{Patched}}^{(n)} = \sum_{i=1}^{\text{|}T_{\text{Test}}\text{|}} 1(\hat{M}_n(x_i) \neq y_i)$$

Additionally, we note that our evaluation method involves random partitions of $T$ and fine-tuning of the target models; the results of both depend on the choice of random seeds. We, therefore, repeated our experiments with 5 different random seeds when partitioning $T$; we fixed the model fine-tuning seed to 42 to prevent evaluation results from being affected by the randomness of training.

5.2.2 Analysis

In Table 3, Figure 2 and Figure 3 of the Appendix, we report the failure rates of TestAug and compare it with existing work (Ribeiro et al., 2020; Tarunesh et al., 2021). The three tasks we study contain 12 capabilities.

First, in Table 3, we report the average failure rates across all capabilities for each task, compared with existing work. We use $T_{\text{TestAug}}$ to represent the test suite by TestAug, and $T_{\text{CheckList}}$ to represent the suite by existing work (Ribeiro et al., 2020; Tarunesh et al., 2021). We use the evaluation methodology in Section 5.2.1 to merge $T_{\text{TestAug}}$ and $T_{\text{CheckList}}$ for comparing their failure rates. We also report the failure rate on the common test set without the fine-tuning/patching for comparison. In addition, we ablate study $T_{\text{TestAug}}$’s performance by removing the cases directly from GPT-3 (i.e., $T_{\text{GPT-3}}$) and the expansion (i.e., $T_{\text{Expansion}}$), using only the resulting subset for patching. We can observe that the resulting failure rate of $T_{\text{TestAug}}$ is consistently lower than $T_{\text{CheckList}}$, indicating that $T_{\text{TestAug}}$ can find more bugs. We also find $T_{\text{TestAug}} \setminus T_{\text{Expansion}}$ to outperform $T_{\text{TestAug}}$ in 4 cases, whereas $T_{\text{TestAug}} \setminus T_{\text{GPT-3}}$ outperforms the latter in only 1 case, this result indicates that GPT-3 generated cases are more important than the expanded cases, whereas the expanded cases are not always helpful. Next, we plot the capability-level failure rates for three tasks (Figure 2, Figure 3 in Appendix). From Figure 2 and Figure 3, we can observe that $T_{\text{TestAug}}$ does not consistently outperform $T_{\text{CheckList}}$ when looking at each linguistic capability; for example, when evaluating sentiment classification task (Figure 2), the ALBERT’s temporal capability gives us a higher failure rate of $\text{FR}_{\text{Patched}}^{(n)}$.

These seeds $\{11, 14, 25, 42, 74\}$ are also randomly generated.
Table 3: The comparison of the bug detection ability between TestAug and CheckList. Each cell shows the failure rate, i.e., the error rate on the held-out validation dataset. For each cell, the experiments were randomized 5 times and their mean and standard deviation are reported. The complete model path of each model can be found from Table 10. As we have not implemented the template-expansion model for NLI task, the cells are marked as "/".

<table>
<thead>
<tr>
<th>Model type</th>
<th>Original</th>
<th>Unpatched</th>
<th>Patched</th>
<th>Patched</th>
<th>Patched</th>
<th>Patched</th>
<th>Patched</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TCheckList</td>
<td>TTestAug</td>
<td>TTestAug | TExpansion</td>
<td>TTestAug | TExpansion</td>
<td>TTestAug | TExpansion</td>
<td>TTestAug | TExpansion</td>
<td></td>
</tr>
<tr>
<td>Sentiment Classification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALBERT</td>
<td>7.3</td>
<td>32.6±5.7</td>
<td>13.4±6.5</td>
<td>11.3±10.0</td>
<td>10.6±6.6</td>
<td>9.6±8.2</td>
<td></td>
</tr>
<tr>
<td>BERTBase</td>
<td>7.6</td>
<td>33.9±6.1</td>
<td>9.0±4.2</td>
<td><strong>8.3±4.2</strong></td>
<td>8.5±1.6</td>
<td>9.9±4.9</td>
<td></td>
</tr>
<tr>
<td>DistillBERT</td>
<td>10.0</td>
<td>29.5±10.9</td>
<td>6.5±3.4</td>
<td><strong>3.9±2.1</strong></td>
<td>4.9±2.1</td>
<td>5.1±3.3</td>
<td></td>
</tr>
<tr>
<td>RoBERTaBase</td>
<td>5.7</td>
<td>14.2±6.1</td>
<td>3.7±2.3</td>
<td>1.6±1.0</td>
<td>2.7±2.7</td>
<td><strong>1.4±1.2</strong></td>
<td></td>
</tr>
<tr>
<td>Paraphrase Detection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALBERT</td>
<td>9.3</td>
<td>38.1±3.8</td>
<td>7.1±0.8</td>
<td>0.6±0.4</td>
<td>5.8±1.8</td>
<td><strong>0.4±0.4</strong></td>
<td></td>
</tr>
<tr>
<td>BERTBase</td>
<td>9.1</td>
<td>36.0±4.9</td>
<td>6.2±1.5</td>
<td>0.5±0.4</td>
<td>5.6±1.1</td>
<td><strong>0.4±0.3</strong></td>
<td></td>
</tr>
<tr>
<td>DistillBERT</td>
<td>10.3</td>
<td>49.8±10.2</td>
<td>12.5±16.4</td>
<td><strong>1.1±2.4</strong></td>
<td>6.4±3.9</td>
<td>7.3±15.8</td>
<td></td>
</tr>
<tr>
<td>Natural Language Inference</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALBERT</td>
<td>9.9</td>
<td>42.8±1.9</td>
<td>30.1±4.2</td>
<td><strong>23.0±1.6</strong></td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>DistillBERT</td>
<td>12.6</td>
<td>34.7±3.6</td>
<td>23.6±6.1</td>
<td><strong>16.5±3.9</strong></td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>RoBERTaLarge</td>
<td>8.1</td>
<td>17.8±4.0</td>
<td>8.3±3.1</td>
<td><strong>8.0±3.1</strong></td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
</tbody>
</table>

$T_{TestAug}$ after patching, indicating our augmented test suite has a weaker bug detection ability; a similar phenomenon could be found when testing vocabulary capability of BERTBase in paraphrase detection task and syntactic and presupposition capability of RoBERTaLarge in natural language inference task. This shows that, despite the overall ability to find more bugs (Table 3), the additional test cases from GPT-3 do not uniformly contribute to the improvement of each specific linguistic capability.

5.3 Evaluating the Diversity of TestAug Results

5.3.1 Metric

As existing approaches rely on manually created templates, they have low linguistic variations. Such issues can be alleviated with the help of GPT-3. In this section, we evaluate the linguistic diversity of the generated test cases. We use two metrics to evaluate the diversity: first, we leverage the Self-BLEU score to evaluate the diversity of an entire test suite; second, to measure the test case-level diversity, we introduce a new metric, i.e., the average number of unique dependency paths in each test case.

Self-BLEU. Self-BLEU is an extension of the regular BLEU that evaluates the diversity of generated texts (Zhu et al., 2018). Given a list of texts $\hat{Y} = \{\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_N\}$, Self-BLEU is the average BLEU score between every single sentence and all other sentences,

$$\text{Self-BLEU}(\hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} \text{BLEU}(\{\hat{Y}_i, \hat{Y}_{\neq i}\})$$

When $k$ is fixed, a lower Self-BLEU score indicates a higher diversity of the sentence.

Number of Unique Dependency Paths. We propose to use the number of unique dependency paths to measure the diversity at the test case level. Each path is a path of POS tags connected by edges in the dependency tree (Jurafsky and Martin, 2000). For example, for the sentence "I (NOUN) love (VERB) chicken (NOUN)" in natural language inference task.

5.3.2 Analysis

We compare the Self-BLEU and the number of unique dependency paths between TestAug and
CheckList in Table 5, where we control the number of test cases under each test\textsuperscript{11}. From Table 5, we can observe that the linguistic diversity of the test suites $T_{\text{GPT-3}}$ show substantial improvement over the template-based counterparts $T_{\text{CheckList}}$: the Self-BLEU4 score has a decrease at least 5.3\% (the natural language inference task) and the number of unique dependency paths is of at least 1.81 times compared to the original test suite (the natural language inference task).

Table 4: TestAug saves manual efforts in generating new test cases and expands the set of available templates.

<table>
<thead>
<tr>
<th>#Uniuqe Seed Sentences</th>
<th>#Unique New Sentences</th>
<th>#Seed Templates</th>
<th>#New Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentiment Classification</td>
<td>292</td>
<td>3275 (11.2x)</td>
<td>29</td>
</tr>
<tr>
<td>Paraphrase Detection</td>
<td>124</td>
<td>6427 (50.4x)</td>
<td>17</td>
</tr>
<tr>
<td>Natural Language Inference</td>
<td>150</td>
<td>4976 (33.2x)</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 5: Linguistic diversity of test suites.

<table>
<thead>
<tr>
<th>Self-BLEU4 (↓)</th>
<th>Number of Unique Dependency Paths (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{T}_{\text{TestAug}}$</td>
<td>0.634</td>
</tr>
<tr>
<td>$\overline{T}_{\text{CheckList}}$</td>
<td>0.853</td>
</tr>
</tbody>
</table>

Paraphrase Detection

| $\overline{T}_{\text{TestAug}}$ | 0.627 | 418 |
| $\overline{T}_{\text{CheckList}}$ | 0.775 | 117 |

Natural Language Inference

| $\overline{T}_{\text{TestAug}}$ | 0.430 | 210 |
| $\overline{T}_{\text{CheckList}}$ | 0.454 | 116 |

5.4 Evaluating the Validity of TestAug Results

In this section, we evaluate the effectiveness of our pipeline of filtering incorrect test cases (i.e., Section 4.2). As is shown in Table 11 of the Appendix, the two-phase approach successfully filters the invalid cases: the classification accuracy is consistently higher than the validity threshold (90\% for sentiment classification and paraphrase detection and 80\% for natural language inference)\textsuperscript{12}. At the same time, the Cohen’s $\kappa$ on test set annotation indicates substantial agreement (McHugh, 2012).

5.5 Evaluation of the Manual Efforts Saved by TestAug

Finally, we quantitatively evaluate the manual efforts saved by TestAug compared to CheckList; the results are reported in Table 4. When querying GPT-3 with a few hundred sentences per task, we obtained a new set of valid test cases 11.2 to 50.4 times in size. The template expansion in turn increased the number of available templates to at least 49.7 times, reducing manual efforts by at least 98.0\% (Table 4)\textsuperscript{13}. This result thus shows that TestAug can largely save manual efforts in creating the test suites.

6 Discussion, Conclusions, and Future Work

This paper introduces a novel framework TestAug for capability-based NLP testing. Addressing current system’s heavy dependence on manual creation of templates, our framework can save at least 98.0\% of the manual annotation effort with GPT-3; meanwhile, the test suites generated with our framework reveal more bugs than existing systems and show better diversity. Our framework guarantees the correctness of the generated cases by removing the invalid output from GPT-3.

The main limitation of TestAug is that GPT-3 fails to generate highly structured test cases, such as cases for extractive question answering. It also struggles to generate cases that require logic or math reasoning. We leave the exploration of these additional capabilities in NLI tasks are below 80\% valid threshold. Automatic filtering these cases out with trained classifiers are left as future work.

\textsuperscript{12}New templates do not cost additional manual efforts when considering all templates as a whole, leading to at least \( \frac{1441 - 29}{1441} \times 100\% = 98.0\% \) saving. With the help of our automatic filtering pipeline, both sentence and template counts could be further increased with additional queries.
cases for future work.
Ethical Considerations

Annotator Rights
Two of the authors (one male and one female; both identified themselves as Asians) annotated the data following annotation guidelines; the guidelines are discussed and finalized after thorough discussions (the violations of these guidelines are discussed in Section 4.2). We acknowledge the annotators’ efforts with shared authorship.

Intended Uses
TestAug’s intended use is as a tool to augment template-based test suites with newly generated test cases from GPT-3; two sets of test cases are then used altogether to evaluate an NLP models’ linguistic capabilities; we believe this application of existing datasets are consistent with their intended uses. We showed the effectiveness of this system in Section 5. We hope the adoption of TestAug into the NLP model development could make newly built NLP models more linguistically capable. Meanwhile, the TestAug includes GPT-3 as a component, we urge users of our system to follow the OpenAI’s usage guidelines.

Potential Misuse
TestAug might be misused to overestimate the models’ linguistic capabilities. Specifically, even though failures on the test suites show models’ shortcomings in a given linguistic capability, the absence of failures does not mean the models being tested are free from bugs; it is likely that test suites are not yet capable enough to reveal the model’s bugs. We, therefore, call for a judicious interpretation of an NLP model’s performance based on TestAug test suites. Moreover, we believe NLP testing is an iterative process; it might take multiple iterations of applying TestAug to reveal the model’s issues in linguistic capabilities.

References

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In EMNLP.


Mor Geva, Yoav Goldberg, and Jonathan Berant. 2019. Are we modeling the task or the annotator? an investigation of annotator bias in natural language understanding datasets. ArXiv, abs/1908.07898.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking nli systems with sentences that require simple lexical inferences. In ACL.


Pratik M. Joshi, Somak Aditya, Aalok Sathe, and Monojit Choudhury. 2020b. Taxinli: Taking a ride up the nlu hill. In CONLL.

Dan Jurafsky and James H. Martin. 2000. Speech and language processing.


14https://beta.openai.com/docs/usage-guidelines


Timo Schick and Hinrich Schütze. 2021. Generating datasets with pretrained language models. In EMNLP.


## Appendix

Table 7: Prompt designs for paraphrase detection and natural language inference tasks.

### Paraphrase Detection

Two sentences are equivalent when using according to:

- {{ Who do analysts think is the smartest footballer in the world? }
  - { Who is the smartest footballer in the world according to analysts? }}
- {{ Who do students think is the top woman in the world? }
  - { Who is the top woman in the world according to students? }}
- {{ Who do readers think is the worst gamer in the world? }
  - { Who is the worst gamer in the world according to readers? }}
- {{ What does the data say about the most popular baby names? }
  - { What are the most popular baby names according to the data? }}

### Natural Language Inference

Write a pair of sentences that have the same relationship as the previous examples. Examples:

- {{ Philip, Charles and Colin are the only children of Henry. }
  - { Henry has exactly 3 children. }}
- {{ Grace, Thomas and Helen are the only children of Andrea. }
  - { Andrea has exactly 3 children. }}
- {{ Don has 2 dollars. He received 8 more dollars. }
  - { Don now has 10 dollars. }}
- {{ Mary has a cat. She also has a dog. }
  - { Mary has two pets. }}

Table 8: Hyperparameter choice for model fine-tuning

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>5e − 6</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
</tr>
<tr>
<td>Number of training epochs</td>
<td>10</td>
</tr>
<tr>
<td>Max. sequence length</td>
<td>128</td>
</tr>
<tr>
<td>Seed</td>
<td>42</td>
</tr>
</tbody>
</table>
Figure 3: Capability-wise error rates on paraphrase detection and natural language inference tasks.
<table>
<thead>
<tr>
<th>Linguistic Capability</th>
<th>Explanation</th>
<th>Template and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sentiment Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negation</td>
<td>Resolving &quot;not&quot;</td>
<td>[neg] {pos_verb_present} {[her]} {air_noun}. I didn’t admire that service.</td>
</tr>
<tr>
<td>SRL</td>
<td>Resolving subjects and objects</td>
<td>Do I think {it} {be} {a:pos_adj} {air_noun}? No</td>
</tr>
<tr>
<td>Temporal</td>
<td>Resolving temporal changes</td>
<td>I {neg_verb_present} this airline, [change] in the past I would {pos_verb_present} it. I regret this airline, although in the past I would appreciate it.</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>Resolving word choice variations</td>
<td>[it] {air_noun} {be} {neg_adj}. That food was ugly.</td>
</tr>
<tr>
<td><strong>Paraphrase Detection</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coref</td>
<td>Resolving male and female names</td>
<td>S1: If {female} and {male} were alone, do you think he would reject her? S2: If {female} and {male} were alone, do you think she would reject him? S1: If Julie and Roy were alone, do you think he would reject her? S2: If Julie and Roy were alone, do you think she would reject him?</td>
</tr>
<tr>
<td>Negation</td>
<td>Resolving &quot;not&quot;</td>
<td>S1: What are things {a:noun} should worry about? S2: What are things that should worry about? S1: What are things a friend should worry about? S2: What are things a friend should not worry about?</td>
</tr>
<tr>
<td>SRL</td>
<td>Resolving subjects and objects</td>
<td>S1: Did [first_name] {verb[0]} the {obj}? S2: Was [first_name] {verb[1]} by the {obj}? S1: Did Sam remember the factory? S2: Was Sam remembered by the factory?</td>
</tr>
<tr>
<td>Taxonomy</td>
<td>Resolving external taxonomic hierarchy</td>
<td>S1: How can I become a[x:1][1] person? S2: How can I become a[x:0][1] person? S1: How can I become a frightened person? S2: How can I become a scared person?</td>
</tr>
<tr>
<td>Temporal</td>
<td>Resolving temporal changes</td>
<td>S1: Is [first_name] [last_name] {a:noun}? S2: Did [first_name] [last_name] use to be {a:noun}? S1: Is Dorothy Clarke an agent? S2: Did Dorothy Clarke use to be an agent?</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>Resolving word choice variations</td>
<td>S1: How can I become less {x[1]}? S2: How can I become more {x[1]}? S1: How can I become less active? S2: How can I become more active?</td>
</tr>
<tr>
<td><strong>Natural Language Inference</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Causal</td>
<td>Resolving actions between one object and two entities</td>
<td>P: {NAME} moved the {OBJECT_TABLE} to {LOCATION_HOUSE1} from {LOCATION_HOUSE2}. H: The {OBJECT_TABLE} is now in {LOCATION_HOUSE1}. P: George moved the notebook to study room from bedroom. H: The notebook is now in study room.</td>
</tr>
<tr>
<td>Lexical</td>
<td>Resolving word choice variations</td>
<td>P: {NAME} was born in {COUNTRY1}. H: {NAME} is born in {COUNTRY2}. P: Emily was born in Germany. H: Emily is born in Malaysia.</td>
</tr>
<tr>
<td>Presupposition</td>
<td>Resolving implications</td>
<td>P: {NAME}’s {T12_RELATION} is {ADJECTIVE_OF_PERSON}. H: {NAME} has a {T12_RELATION}. P: Florence’s brother is intolerant. H: Florence has a brother.</td>
</tr>
<tr>
<td>Quantifier</td>
<td>Resolving &quot;all&quot; (universal quantification) and &quot;some&quot;, &quot;none&quot; (existential quantification)</td>
<td>P: None of the {OBJECTS} are {COLOUR1} in colour. H: Some of the {OBJECTS} are {COLOUR2} in colour. P: None of the cars are maroon in colour. H: Some of the cars are pink in colour.</td>
</tr>
<tr>
<td>Syntactic</td>
<td>Resolving ellipsis</td>
<td>P: {NAME} tried but wasn’t able to {VERB}. H: {NAME} didn’t try to {VERB}. P: Alan tried but wasn’t able to give. H: Alan didn’t try to give.</td>
</tr>
<tr>
<td>World</td>
<td>Resolving world knowledge such as geography</td>
<td>P: {NAME} lives in {T4_CAPITAL1}. H: {NAME} lives in {T4_COUNTRY2}. P: Ken lives in Kathmandu. H: Ken lives in North Korea.</td>
</tr>
</tbody>
</table>
Figure 4: The command line interface for data annotation. Annotators are given a test and three associated test cases from the template-based test suite; they are asked to annotate the validity of the GPT-3-generated sentences. Annotators are reminded of the guidelines for filtering invalid samples when labeling each sentence (shown at the top of the interface). We communicated explicitly for the intended uses of the annotated datasets before the annotation.

Table 10: The fine-tuned models we evaluated in this paper.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Task</th>
<th>Size</th>
<th>Checkpoint Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistillBERT</td>
<td>Sentiment Analysis</td>
<td>Small</td>
<td>textattack/distilbert-base-cased-SST-2</td>
</tr>
<tr>
<td>ALBERT</td>
<td>Sentiment Analysis</td>
<td>Small</td>
<td>textattack/albert-base-v2-SST-2</td>
</tr>
<tr>
<td>BERT_base</td>
<td>Sentiment Analysis</td>
<td>Base</td>
<td>textattack/bert-base-uncased-SST-2</td>
</tr>
<tr>
<td>RoBERTa_base</td>
<td>Sentiment Analysis</td>
<td>Base</td>
<td>textattack/roberta-base-SST-2</td>
</tr>
<tr>
<td>DistillBERT</td>
<td>Paraphrase Detection</td>
<td>Small</td>
<td>textattack/distilbert-base-cased-QQP</td>
</tr>
<tr>
<td>ALBERT</td>
<td>Paraphrase Detection</td>
<td>Small</td>
<td>textattack/albert-base-v2-QQP</td>
</tr>
<tr>
<td>BERT_base</td>
<td>Paraphrase Detection</td>
<td>Base</td>
<td>textattack/bert-base-uncased-QQP</td>
</tr>
<tr>
<td>DistillBERT</td>
<td>Natural Language Inference</td>
<td>Small</td>
<td>textattack/distilbert-base-cased-snli</td>
</tr>
<tr>
<td>ALBERT</td>
<td>Natural Language Inference</td>
<td>Small</td>
<td>textattack/albert-base-v2-snli</td>
</tr>
<tr>
<td>RoBERTa_large</td>
<td>Natural Language Inference</td>
<td>Large</td>
<td>ynie/roberta-large-snli_mnli_fever_anli_R1_R2_R3-nli</td>
</tr>
</tbody>
</table>
Table 11: Proportion of valid sentences and performance of trained classifiers for automatic filtering.

<table>
<thead>
<tr>
<th>Linguistic Capability</th>
<th>Test</th>
<th>Accuracy</th>
<th>F1</th>
<th>Proportion of Valid Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sentiment Analysis</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRL</td>
<td>A negative sentiment sentence with negative sentiment question and word yes as the answer.</td>
<td>/</td>
<td>/</td>
<td>1.000</td>
</tr>
<tr>
<td>SRL</td>
<td>A positive sentiment sentence with positive sentiment question and word yes as the answer.</td>
<td>/</td>
<td>/</td>
<td>0.994</td>
</tr>
<tr>
<td>SRL</td>
<td>My opinion is more important than others’ when expressing positive sentiment.</td>
<td>/</td>
<td>/</td>
<td>0.948</td>
</tr>
<tr>
<td>SRL</td>
<td>A negative sentiment sentence with positive sentiment question and word no as the answer.</td>
<td>/</td>
<td>/</td>
<td>0.928</td>
</tr>
<tr>
<td>Temporal</td>
<td>I used to have negative sentiment to something, but now I have positive sentiment to it.</td>
<td>/</td>
<td>/</td>
<td>0.922</td>
</tr>
<tr>
<td>Negation</td>
<td>A negative sentiment sentence with negated positive word.</td>
<td>/</td>
<td>/</td>
<td>0.910</td>
</tr>
<tr>
<td>SRL</td>
<td>My opinion is more important than others’ when expressing negative sentiment.</td>
<td>/</td>
<td>/</td>
<td>0.900</td>
</tr>
<tr>
<td>Temporal</td>
<td>I used to have positive sentiment to something, but now I have negative sentiment to it.</td>
<td>0.942</td>
<td>0.967</td>
<td>0.873</td>
</tr>
<tr>
<td>Negation</td>
<td>I thought something was positive, but it was negative.</td>
<td>0.934</td>
<td>0.961</td>
<td>0.860</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>A negative sentiment sentence with negative words.</td>
<td>0.897</td>
<td>0.932</td>
<td>0.804</td>
</tr>
<tr>
<td>Negation</td>
<td>A negative sentiment sentence with negated positive sentiment word and neutral contents in the middle.</td>
<td>0.915</td>
<td>0.935</td>
<td>0.653</td>
</tr>
<tr>
<td><strong>Paraphrase Detection</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>Two sentences are different when talking about a person’s current job and his or her previous job.</td>
<td>/</td>
<td>/</td>
<td>1.000</td>
</tr>
<tr>
<td>Negation</td>
<td>Two sentences are different when talking about someone should and should not do something.</td>
<td>/</td>
<td>/</td>
<td>0.973</td>
</tr>
<tr>
<td>Temporal</td>
<td>Two sentences are different when talking about a person’s current job and his or her future job.</td>
<td>/</td>
<td>/</td>
<td>0.964</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>Two sentences are different when adjectives are modified by more and less.</td>
<td>/</td>
<td>/</td>
<td>0.912</td>
</tr>
<tr>
<td>Taxonomy</td>
<td>Two sentences are equivalent when the nouns are modified by synonymous adjectives.</td>
<td>0.889</td>
<td>0.933</td>
<td>0.885</td>
</tr>
<tr>
<td>Coref</td>
<td>Two sentences are different when swapping the subjects and objects.</td>
<td>0.980</td>
<td>0.989</td>
<td>0.843</td>
</tr>
<tr>
<td>Temporal</td>
<td>Two sentences are different when talking about doing something before and after another thing.</td>
<td>0.982</td>
<td>0.989</td>
<td>0.800</td>
</tr>
<tr>
<td>Temporal</td>
<td>Two sentences are different when describing doing something before and after some specific time.</td>
<td>0.982</td>
<td>0.988</td>
<td>0.782</td>
</tr>
<tr>
<td>Negation</td>
<td>Two sentences are different when talking about the properties of doing or not doing something.</td>
<td>0.965</td>
<td>0.977</td>
<td>0.754</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>A sentence with the noun modified by an adjective is equivalent to the sentence without adjective.</td>
<td>0.920</td>
<td>0.946</td>
<td>0.740</td>
</tr>
<tr>
<td>SRL</td>
<td>Two sentences are equivalent when using according to.</td>
<td>0.958</td>
<td>0.971</td>
<td>0.708</td>
</tr>
<tr>
<td>Negation</td>
<td>Two sentences are different when describing a person with adjective and a clause including the negation of the same adjective.</td>
<td>0.871</td>
<td>0.913</td>
<td>0.690</td>
</tr>
<tr>
<td>Taxonomy</td>
<td>Two sentences are equivalent when one has an adjective modified by more and the other one has an antonym modified by less.</td>
<td>0.920</td>
<td>0.909</td>
<td>0.511</td>
</tr>
<tr>
<td>Coref</td>
<td>Two sentences are different when referring someone’s family using different pronouns.</td>
<td>0.940</td>
<td>0.940</td>
<td>0.500</td>
</tr>
<tr>
<td>SRL</td>
<td>Two sentences are different when swapping active and passive action.</td>
<td>0.930</td>
<td>0.933</td>
<td>0.471</td>
</tr>
<tr>
<td><strong>Natural Language Inference</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lexical</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.975</td>
</tr>
<tr>
<td>Syntactic</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.936</td>
</tr>
<tr>
<td>Presupposition</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.903</td>
</tr>
<tr>
<td>World</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.902</td>
</tr>
<tr>
<td>Quantifier</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.895</td>
</tr>
<tr>
<td>Causal</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.857</td>
</tr>
<tr>
<td>Conditional</td>
<td></td>
<td>/</td>
<td>/</td>
<td>0.834</td>
</tr>
</tbody>
</table>

Table 12: Creating new templates based on test cases generated by GPT-3.

<table>
<thead>
<tr>
<th>Original Template and Test Case</th>
<th>Generated Test Case</th>
<th>New Template</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sentiment Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No one {pos_verb_present}s {the} {air_noun}. No one enjoys that seat.</td>
<td>This is not an easy service to appreciate. That customer service was not fun. I don’t think your customer service is admired.</td>
<td>This is not an easy [air_noun] to appreciate. That customer [air_noun] was not fun. I don’t think your customer [air_noun] is admired.</td>
</tr>
<tr>
<td>Is it {mid} to {activity} before {hour} [ampm]?</td>
<td>Is it bad to drink before 8pm</td>
<td>Is it bad to {activity} before 8pm</td>
</tr>
<tr>
<td>Is it {mid} to {activity} after {hour} [ampm]?</td>
<td>Is it bad to drink after 8pm</td>
<td>Is it bad to {activity} after 8pm</td>
</tr>
<tr>
<td>Is it healthy to drink before 10am?</td>
<td>Is it acceptable to drink before 2pm</td>
<td>Is it {mid} to {activity} before 2pm</td>
</tr>
<tr>
<td>Is it healthy to drink after 10am?</td>
<td>Is it acceptable to drink after 2pm</td>
<td>Is it {mid} to {activity} after 2pm</td>
</tr>
<tr>
<td>Is it advisable to eat before 8pm</td>
<td>Is it advisable to eat before 8pm</td>
<td>Is it advisable to {activity} before 8pm</td>
</tr>
<tr>
<td>Is it advisable to eat after 8pm</td>
<td>Is it advisable to eat after 8pm</td>
<td>Is it advisable to {activity} after 8pm</td>
</tr>
<tr>
<td><strong>Paraphrase Detection</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is it {mid} to {activity}</td>
<td>Is it bad to drink before 8pm</td>
<td>Is it bad to {activity} before 8pm</td>
</tr>
<tr>
<td>Is it {mid} to {activity} after {hour}</td>
<td>Is it bad to drink after 8pm</td>
<td>Is it bad to {activity} after 8pm</td>
</tr>
<tr>
<td>Is it healthy to drink before 10am?</td>
<td>Is it acceptable to drink before 2pm</td>
<td>Is it {mid} to {activity} before 2pm</td>
</tr>
<tr>
<td>Is it healthy to drink after 10am?</td>
<td>Is it acceptable to drink after 2pm</td>
<td>Is it {mid} to {activity} after 2pm</td>
</tr>
<tr>
<td>Is it advisable to eat before 8pm</td>
<td>Is it advisable to eat before 8pm</td>
<td>Is it advisable to {activity} before 8pm</td>
</tr>
<tr>
<td>Is it advisable to eat after 8pm</td>
<td>Is it advisable to eat after 8pm</td>
<td>Is it advisable to {activity} after 8pm</td>
</tr>
</tbody>
</table>