A Case Study of Efficacy and Challenges in Practical Human-in-Loop Evaluation of NLP Systems using Checklist

Shaily Bhatt[1] **Rahul Jain**[2] **Sandipan Dandapat**[2] **Sunayana Sitaram**[1]
[1] Microsoft Research, Bangalore, India
[2] Microsoft R&D, Hyderabad, India
{t-shbhatt, jain.rahul, sadandap, sunayana.sitaram}@microsoft.com

Abstract

Despite state-of-the-art performance, NLP systems can be fragile in real-world situations. This is often due to insufficient understanding of the capabilities and limitations of models and the heavy reliance on standard evaluation benchmarks. Research into non-standard evaluation to mitigate this brittleness is gaining increasing attention. Notably, the behavioral testing principle 'Checklist', which decouples testing from implementation revealed significant failures in state-of-the-art models for multiple tasks. In this paper, we present a case study of using Checklist in a practical scenario. We conduct experiments for evaluating an offensive content detection system and use a data augmentation technique for improving the model using insights from Checklist. We lay out the challenges and open questions based on our observations of using Checklist for human-in-loop evaluation and improvement of NLP systems. *Disclaimer: The paper contains examples of content with offensive language. The examples do not represent the views of the authors or their employers towards any person(s), group(s), practice(s), or entity/entities.*

1 Introduction

NLP systems have been known to learn spurious patterns from data to achieve high accuracy on test sets (Goyal et al., 2017; Gururangan et al., 2018; Glockner et al., 2018; Tsuchiya, 2018; Geva et al., 2019). Evaluating models on static benchmarks and on test sets that have a similar distribution to the training data has resulted in an overestimation of model performance (Belinkov and Bisk, 2018; Recht et al., 2019) and models becoming increasingly fragile or less useful in real-world settings. This can be due to various factors such as language complexity and variability, the difference between training, testing, and real-world data, and insufficient understanding of the capabilities and limitations of the model itself. When deployed in the wild, such systems tend to break down, resulting in grossly incorrect predictions. This leads to mistrust in the system at two levels – first, on individual predictions and second, on the system's soundness in uncontrolled environments such as usage after deployment (Ribeiro et al., 2016).

Further, evaluation benchmarks are also becoming increasingly obsolete due to the exponential rise in data and compute-heavy systems that exceed performance expectations, bringing the benchmark's 'toughness' and hence, its reliability into question (Nie et al., 2020). In order to mitigate this limitation of static evaluation, several approaches are used to evaluate other model aspects including, but not limited to, robustness (Rychalska et al., 2019), fairness (Prabhakaran et al., 2019), consistency (Ribeiro et al., 2019), explanations (Ribeiro et al., 2016), and adversarial performance (Ribeiro et al., 2018b; Iyyer et al., 2018; Nie et al., 2020).

Human-in-Loop processes can be used to complement the capabilities of automation with human expertise (Ribeiro et al., 2020; Potts et al., 2020; Nie et al., 2020; Ribeiro et al., 2018b). Previous studies have shown that using humans to close the loop of the process of evaluation, explanation, or improvement can lead to a much better understanding of the system through higher explainability (Ribeiro et al., 2018a, 2016), better detection of model failures (Ribeiro et al., 2020; Iyyer et al., 2018), and easier bug-fixing (Ribeiro et al., 2020, 2018b), resulting in robustness of the model in practical scenarios and increased trust in its predictions.

Ribeiro et al. (2020) introduced a behavioral testing strategy that decouples testing from model implementation. Using human-generated test sets, they showed that state-of-the-art NLP models for multiple tasks fail to perform well for basic capabilities. We describe the framework in detail in section 2.

In this paper, we describe a case study of using the Checklist paradigm of evaluation in a practical scenario. Specifically, we used Checklist to evaluate and debug an offensive content detection system. We found that Checklist can lead to effective pinpointing of specific capabilities for which the model, despite impressive performance on a standard benchmark test set, failed. Further, these insights can be used to improve the model using targeted data augmentation to debug specific model failures. However, we found that using Checklist for evaluation and improvement is not always foolproof. We discuss the challenges and open questions observed during these experiments.

The rest of the paper is organized as follows: First, we give a brief overview of the Checklist framework. In Section 3, we describe our case study, including the capabilities we test, results on the base model, the method applied to use these insights for improvement, and the results thereafter. Finally, we present a detailed analysis of some of the most imminent challenges with using Checklist for our experiments.

2 Overview of Checklist

The Checklist framework (Ribeiro et al., 2020) introduces a human-in-loop behavioral testing technique for evaluating NLP systems. The authors argue that even though models perform well on static benchmarks, they fail to perform in real-world scenarios for basic capabilities. They release an open-source package [1] with functionality to create template sets and run software engineering-like decoupled testing on black-box models.

The individual phenomena tested using Checklist are known as capabilities. These capabilities are based on model expectations and the language usage that it needs to handle. For example, Negation is a capability of a Sentiment Analysis model - the model should be able to distinguish 'happy' and 'not happy' as two opposite sentiments despite the overlapping word 'happy'.

The Checklist framework provides three different test types. The Minimum Functionality Tests (MFTs) are simple tests, similar to unit tests in software testing, that can test predictions on specific model capabilities. Most of the capabilities tested in our case study are MFTs. Invariance tests (INVs) are a test type where small semantic-preserving perturbations are applied to the test cases, and it is expected that the model output should not change. For example - in our case, while testing for the Robustness of the model, small typos are introduced. Directional Expectation tests (DIRs) are similar to INV, except that the model output is expected to change in a certain way.

In order for humans to generate test cases, Checklist uses Templates and Lexicons. For example: 'I {POSITIVE_VERB} {ACTIVITY}.' is a template. 'POSITIVE_VERB' and 'ACTIVITY' in this template are two different keywords in the lexicon, each taking a specific set of values. For example, POSITIVE_VERB = ['like', 'love', 'enjoy'] and ACTIVITY = ['dancing', 'hiking' 'cooking', 'coding']. The template generates 12 examples - the Cartesian product of the values of the two lexicon keywords in the template.

In the original Checklist work, the authors test state-of-the-art and commercial systems across three tasks revealing unprecedented failure rates even for the most basic capabilities. For more detailed information on these results, we refer the reader to the original paper.

3 A Case Study of Using Checklist

We used Checklist to evaluate and improve an in-house offensive content detection system. While we studies a particular system as a use case, the process can be applied to any NLP model.

The first step was to create a documentation of expectations. This documentation was important to lay out guidelines for what constitutes positive or negative examples in specific settings. In accordance with standard text classification annotation, for this task, offensive content is the positive class, and non-offensive content is the negative class. The documentation or guidelines contained concrete definitions of various offensive content categories that the model is expected to detect. These are henceforth known as the model capabilities and are described in section 3.1. This is important for Checklist evaluation because it relies on building templates that target specific model capabilities. We gave this documentation to the annotators who were expected to use these guidelines to come up with templates of specific capabilities that could be used to find bugs.

Since offensive classification is a task that inherently has a class imbalance with fewer examples in the positive class in most publicly available datasets (de Gibert et al., 2018; Davidson et al., 2017), including ours, we asked the annotators to develop templates that would result in offensive data. An-

[1] https://github.com/marcotcr/checklist

other reason for this design decision was that while offensive content is more likely to follow a certain pattern - a combination of entities and offensive language, non-offensive content would not follow any discernible patterns making it difficult to convert it into templates. Further, from an application perspective in real-world settings, where such a model is employed to flag or filter offensive content, a higher recall is preferable, as false negatives (examples that are offensive but predicted as non-offensive) need to be avoided even at the cost of a few false positives.

3.1 Capabilities

Based on the documentation the capabilities tested for detection of offensive content were:

1. **Characterization**: Derogatory content/Defamation targeted towards individuals or groups (people who share certain attributes like sex, race, religion, nationality, occupation, age, etc.) including stereotypes. Also includes positive and negative characterizations of individuals or groups including comments on sexual orientation, ethics, morality, habits, physical appearance, or other characteristics.
   (a) **Simple Characterization of Individuals**: Characterizing individuals. *Example*: *Trump is an idiot, is Joe Biden racist?, Kevin Spacey gay, Fat Meghan Markle, Bin Laden was amazing*, etc.
   (b) **Simple Characterization of Groups**: Characterizing groups *Example*: *Mexicans are lazy, Are all whites racist, Jews are greedy, females belong in the kitchen, all men are pigs, people from middle east stink*, etc.
2. **Negation**: Negation of characteristics targeted towards individuals or groups. Also includes negation of positive and negative characterizations of individuals or groups including comments on sexual orientation, ethics, morality, habits, physical appearance, or other characteristics.
   (a) **Negation of Positive characterization of Individuals**: Negating a positive characterization of any individual. Also includes content that seeks or suggest negation of rights or equality to specific entities. *Example*: *Trump is not smart, Justin Beiber is not an ethical person, Gandhi was not honest*, etc.
   (b) **Negation of Positive Characterization of Groups**: Negating positive characterization of groups. Also includes content that seeks or suggest negation of rights or equality to specific groups. *Example*: *Females are not smart, Muslims are not honest, Policemen cannot be good, Women should not be allowed to go to college*, etc.
   (c) **Negation of Negative Characterization of Individuals**: Negating negative characterization of individuals. *Example*: *Bin Laden was not bad, Hitler was not brutal*, etc.
   (d) **Negation of Negative Characterizations of Groups**: Negating negative characterization of groups. *Example*: *Jihadists are not bad, Racist people are not awful*, etc
3. **Violence** Explicitly seeking or suggesting violence against Groups or Individuals. Also includes content that in general talks about violent actions in an offensive fashion but is not directed towards any specific entity. *Example: ISIS beheadings, Trump should be assassinated, Should men hit women, North korea should be nuked*, etc.
4. **Unsafe**: Seeking or suggesting unsafe content related to topics including, but not limited to, suicide, self-harm, illegal drugs, addictive or harmful medication, etc. *Example: Easy ways to kill oneself, Where to buy levothyroxine without prescription, How to make meth at home*, etc.
5. **Racy**: Sexually suggestive content which may or may not be targeted to individuals or groups. *Example*: *Pictures of hot women swimmers, Japanese men in bedroom*, etc
6. **Robustness**: To test if the system is robust to minor perturbations like typos or punctuation that are consistent with intentional or unintentional usage in writing offensive content.
   (a) **One Typo**: Introducing one random typo in the example by flipping adjacent characters. *Example*: *fcuk*.
   (b) **Two Typo**: Introducing one random typo in the example by flipping adjacent characters. *Example*: *All mne ar epigs*.
   (c) **Punctuation Insertion**: Random inser-

tion of punctuation marks in the example. *Example*: *f*ck*.

3.2 Using Checklist to Evaluate the Base Model

We used one template set and documented the failure rates of an in-house XLMR-based offensive content detection model, henceforth referred to as the base model. This base model had a good performance on the standard static benchmark test sets of 10k instances,[2] similar to other state-of-the-art systems for offensive content detection. The static test set had a similar distribution to the training data and has roughly 2.5k positive and 7.5k negative examples. The metrics on the static test set are available in the base model row of Table 1. The model was a 24-layer transformer-based XLM-R model fine-tuned with 481k examples of offensive and hate speech data, out of which 198k were positive examples. The model was trained with a learning rate of 5e-6 for 10 epochs with a batch size of 128.

Targeting specific capabilities using Checklist showed huge failure rates, indicating that the model still failed to meet the expectations even with good performance metrics. This is consistent with the original findings of Ribeiro et al. (2020), where multiple state-of-the-art models were found to have huge failure rates for the many basic capabilities.

Particularly in derogatory content, offensive content against a specific person seemed to be tougher to detect as compared to offensive content against a group. This may be because the names used to generate the test cases for offensive content against a person may not necessarily be names of famous people seen in training data, and the model was unable to generalize offensive language detection to unseen named entities. It is also possible that offensive content against specific groups is a more sensitive issue and is thus represented more in the base model's training data. Further, the model was unable to handle negation very well. This is consistent with the findings of Ribeiro et al. (2020) who also found that state-of-the-art sentiment analysis models failed much more when dealing with negation.

In capabilities of unsafe and violent content, the failure rates were comparatively lower. This can be attributed to the fact that such content is more likely to contain specific keywords or patterns that the model has learned to classify as offensive during training, resulting in lower failure rates even when tested using templates. In Racy content, however, this might have been tougher. This is because racy content is often observed to be multi-intentioned. The same content can be an innocent statement or a racy statement. For example, words like 'cock' or 'chicks' that are often used in an explicit or racy sense and can also refer to their actual (non-racy) meaning.

Finally, Checklist evaluation revealed that the model was NOT robust to minor perturbations. This is an important finding because it is expected that the model would come across content that the user intentionally or unintentionally mistypes. However, such perturbations may not have been reflected in the training and standardized testing data.

As seen in the Base Model row of Table 2, the failure rates for examples generated by templates of specific capabilities is high.

Model	Precision	Recall	F1 Score
Base	79.75	80.13	79.94
Aug-1	80.20	79.30	79.75
Aug-2	79.27	80.13	79.70
Aug-3	79.50	80.13	79.81
Aug-4	80.56	80.25	80.40
Aug-5	80.14	80.13	80.14

Table 1: Metrics on static benchmark test set

3.3 Improving the Model

So far, consistent with previous results, Checklist evaluation revealed important gaps in the base model. The interesting follow-up question is how to use these insights to improve the model to overcome current limitations. Typically, in deep learning models, model improvements result from improved model architectures, better training or fine-tuning strategies, or more data. However, these strategies do not directly address the limitation of models in specific capabilities in the way that Checklist reveals. Thus, we explored the use of insights from the human-in-loop evaluation that can help improve the model in these specific settings while also testing the improved model against standard benchmarks used to evaluate the model.

3.3.1 Data Augmentation Methodology

We used an iterative process of data augmentation to improve the model using the insights of model failures from Checklist evaluation and data generated by templates. We chose this iterative data augmentation method due to its demonstrated

[2]Created by three human judges with majority voting

	1.a	1.b	2.a	2.b	2.c	2.d	3	4	5	6.a	6.b	6.c
Base	47.37	22.84	32.13	36.73	37.77	46.16	18.70	6	44.70	46.62	59.11	42.61
Aug-1	22.21	0.05	0.03	0	1.07	0.02	0	0	0	3.54	4.23	1.08
Total Examples	10k	5.5k	3k	6k	7.5k	5.5k	4.3k	1k	2.7k	12.3k	12.3k	12.7k

Table 2: Failure Rates (%) of Base and Improved models for Test holdout set for different capabilities 3.1

Model	**Training Data**	**TS-1**	**TS-2**	**TS-3**	**TS-4**	**TS-5**
Base	Base training data (BTD)	38.0 (0)	20.05 (0)	28.24 (0)	26.35 (0)	34.14 (0)
Aug-1	BTD + data from TS-1	4.0* (-34)	12.67 (-7.38)	5.66 (-22.58)	18.83 (-7.52)	13.67 (-20.47)
Aug-2	BTD + data from TS-2	31.73 (-6.27)	0.01* (-20.04)	19.34 (-8.9)	13.28 (-13.28)	26.97 (-7.17)
Aug-3	BTD + data from TS-3	30.21 (-7.79)	15.53 (-4.52)	0.01* (-28.23)	26.11 (-0.24)	23.57 (-10.57)
Aug-4	BTD + data from TS-4	34.29 (-3.71)	9.61 (-10.44)	24.98 (-3.26)	0* (-26.35)	30.29 (-3.85)
Aug-5	BTD + data from TS-5	32.18 (-5.82)	15.74 (-4.31)	19.34 (-8.9)	19.35 (-7.0)	0.01* (-34.13)

Table 3: Average (weighted across different capabilities by number of examples in each) failure rates (%) of different models on independently created template sets. Figures in bracket show change in failure rate from the failure rate of base model tested on the particular template set (* refers to tested on the test holdout set such that the testing examples are disjoint from training data of the augmented model but come from the same template set)

success in improving NLI models by Nie et al. (2020), who proposed iterative Human-And-Model-in-the-Loop-Enabled-Training (HAMLET) to create dynamic and harder adversarial test sets for that 'fools'[3] the model. These harder examples are then used to re-train the model, and the process is repeated. Potts et al. (2020) also successfully use a similar human-in-loop feedback process with data augmentation to create iterations of datasets and better models for sentiment analysis.

Our process in spirit is similar, except instead of adversarial examples, we focused on specific capabilities from the Checklist evaluation using templates. A set of examples generated from the same Checklist templates, which is disjoint from the test examples themselves, were appended to the model's original training set, and the model was re-trained. This yielded a new model, henceforth called the augmented model. The augmented model was then tested on the set of examples that was earlier used to test the base model.

Specifically, TS-1 was the set of templates used to test the base model. This template set was generated by a human annotator, known to have sufficient expertise of English. The data generated from the TS-1 was divided into a training subset (TrS) and test subset (TeS) with a ratio of 60:40. First, the base model's failure rates on TeS were recorded as shown in Table 2. Now the TrS was combined with the base model's training data, and the model was re-trained. This re-trained model is called the augmented model. The data from TeS was now used to test this augmented model for the capabilities captured in TS-1.

[3]flips the output of the model

3.3.2 Performance After Data Augmentation

We found that the failure rates of the augmented model dropped significantly. Interestingly, the performance on the static evaluation test sets neither improved nor degraded substantially, which can be seen in Table 1. Here, Aug-1 was the model obtained by retraining the base model with the original data plus data from TrS of TS-1. The rest of the four augmented models, Aug 2-5, will be described subsequently. This shows that while data augmentation helps specific capabilities, it does not degrade performance on the static benchmark leading to the conclusion that the retrained model is not over-fitted to the examples generated using Checklist.

The fact that the performance on the benchmark test set did not improve showed that static benchmark evaluation sets failed to evaluate the model rigorously enough for important capabilities. Adding data points that make the model more robust to such examples improves the model overall. However, this improvement was not captured in the static evaluation as the test set might not have contained such specific examples for these capabilities in the first place. This is why the failure of model in these scenarios went unnoticed till it was evaluated specifically for those capabilities using Checklist. This observation bolsters the case of using Checklist evaluation for better understanding and explainability of the limitations of the model.

3.3.3 Testing on Multiple Template Sets

The fact that TrS and TeS have very similar (but not the same) examples as they were generated from the same set of templates can be one reason for the augmented model's extremely low failure rate. However, to analyze whether the model learned generalizable capabilities from the TS-1 and ascertain that these gains in performance corresponding to lower failure rates are not specific to a template set, we asked a new independent annotator to use the documentation guidelines to create templates from scratch. The data generated from this independently generated template set is used to evaluate the base and augmented model (Aug-1, which was trained on data from TS-1). This process was carried out with four different annotators, resulting in 4 new augmented models (Aug 2-5).

Specifically, we created a larger study and asked four more annotators to create template sets independently using the documentation guidelines. These template sets are called TS-2, TS-3, TS-4, and TS-5. Data from each template set was also split into training and testing sets with a ratio of 60:40. The same base model was first used, and failure rates were recorded on each template set. Next, four more augmented models were created (Aug-2, Aug-3, Aug-4, and Aug-5). For creating Aug-i ($2 \leq i \leq 5$), the training data from TS-i was combined with the base model's training data, and the model was re-trained on this entire dataset. Now the failure rates of Aug-i were recorded on the held out test set (data points coming from the same templates but disjoint from the training augmented data) of TS-i and the entire data from the rest of the template sets.

All template sets were generated by annotators with expertise of English language and were cross-checked for correctness. The number of templates in each template sets ranged from 18-25 distributed among the different capabilities. The number of examples generated from template sets that were added to retraining of the model (including perturbed examples for robustness test) were close to 50k for each of the augmented models. There were no templates that were exactly the same in any pairs of template sets, though, there were some templates that were similar. The overlap in terms of examples generated was less than 0.02% between any of the sets. The lexicon keywords had some common vocabulary. However, this can be expected due to the specificity of the task and the words that are commonly used in such offensive statements.

3.3.4 Performance on Multiple Template Sets

We report the average failure rates in Table 3. The reported average is the weighted average of failure rates across different capabilities, weighted according to the number of examples the template set has for that capability. The results across template sets vary, and we discuss the challenge of ascertaining template quality in detail in section 4.3.

We found that for all our augmented models, the failure rates between the base and augmented model significantly differed for the test holdout of its own template set. Further, the augmented models showed better performance than the base model across all examples from all other template sets that were generated independently by different annotators. In fact, we saw improvements up to 15-20% in multiple cases (e.g. Aug-1 on Ts-3 and Ts-5) . This indicates that the model did learn some generalizable capabilities irrespective of the template set used for augmentation.

Grouping the results by capability, we found a general trend of lower failure rates in augmented models. There were no clear trends of a particular capability consistently benefitting more or less. The failure rates of Template Sets 2-5 on the Augmented models 2-5 and base model grouped by capabilities are in the Appendix.

4 Challenges and Open Questions

Our case study was an experiment of using Checklist to debug NLP systems. It presents optimistic findings for using human-in-loop for improving model performance. However, using this technique for evaluation and improvement is not straightforward or foolproof. In this section, we discuss some nuances and challenges that we observed while conducting these experiments.

4.1 Resource Requirement

The process, while effective, is intensive in both human and computational resources.

Generating templates from scratch required a significant amount of annotator hours. In our experiments, it took 1 hour to create 5-7 templates spanning 1-2 capabilities. This time can vary from person to person. A single annotator required a minimum of half of a workday[4] and a maximum of 2 workdays to come up with template sets. The

[4]a workday is taken as 8 hours

time may also vary based on the task for which templates are being generated.

Once the template sets are created, generating Checklist reports for evaluation is computationally cheap, the cost of model inference notwithstanding. However, using the insights of this evaluation to carry out the targeted data augmentation procedure can be compute-heavy. Retraining the model can cost significant time, money, and energy. Fine-tuning, though computationally cheaper *can* lead to over-fitting on template sets, which is why we chose not to take the approach. Further, going through the iterative and parallel versions of the process would require further investment of human and computational resources to generate more template sets, employ more annotators, and repeated retraining of the model.

4.2 Methods to Improve the Model

In the current version of the process, we used a simple but effective iterative data augmentation procedure. While this is effective in our case it *can* lead to over-fitting or catastrophic forgetting in deep learning models. Furthermore, as stated earlier, the process itself is compute-expensive.

Data augmentation may not be the only (or the most optimal) solution. Some other methods that can be utilized are continual training or fine-tuning. Furthermore, there can be more than one way to combine the initial training and template generated datasets to yield better performance. Thus, the effective use of the insights from Checklist evaluation still remains an open question for future studies.

4.3 Template and Template Set Quality

An important question for any evaluation technique, whether static benchmark or human-generated templates, is its quality. In both cases, it is difficult to quantify quality.

The main reason why it is important to estimate the quality of template sets is evident from the results of Table 3. None of the augmented models are better for all the template sets across the board, and performance on the same template set can vary significantly for different augmented models created by augmenting different data points. Thus, the templates that humans come up with and the examples that those templates generate can significantly impact how much the model improves.

Quality can be viewed in two ways, absolute quality, and relative quality. Absolute quality of a template refers to easily quantifiable measures such as the number of examples it generates and the capabilities it covers. On the other hand, two templates are compared for their quality in the case of relative quality. In this case, the higher quality template would intuitively be one that can find more bugs or result in higher failure rates in the model. It is important to note that higher absolute quality may not always result in higher relative quality. A template can generate more examples and cover more capabilities and give low failure rates, leading to finding fewer bugs than another template that generates fewer examples or spans fewer capabilities.

Relative template quality is a more effective way for quality analysis of templates because it is driven by failure rates of the model on the template compared to other templates, and this is the basis of finding bugs using Checklist. However, whether a template is 'tougher' (hence, of higher quality with respect to relative quality evaluation) or 'easier' is subjective to the model and its training data. In other words, a template that results in higher failure rates for a particular model as compared to another template of the same capability can show lower failure rates when used with another model and vice versa. Furthermore, human analysis of template quality may not always sync with the model performance. That is, a template that a human may deem to be 'tougher' for a model may not be so.

Following the definition of template quality for individual templates may not always extrapolate to a template set's quality. That is, while comparing the quality of two template sets, it can be possible (and in fact, often observed in our study) that a template set may contain some templates that are of a higher relative quality and some templates that are of lower relative quality as compared to the templates of another template set. This makes it even more difficult to quantify even the relative quality of template sets that span multiple templates and capabilities.

Further, template generation by humans is an endless process; one can keep on generating more and more templates given time. In fact, in the extreme situation, it is possible that the an iteration of Checklist evaluation may not reveal any actionable bugs, in such a scenario, it would be unclear as to how many iterations would be needed in order to claim that the model does not have any bugs.

Moreover, within template sets, multiple capabilities are covered by putting together different templates by annotators. Our results show that this does not lead to consistent improvements across capabilities. Thus, obtaining the best combination of different templates is not straight-forward. A detailed study into what constitutes better quality templates can help ascertain a more effective selection process from a large set.

Finally, multiple templates can be combined to form template sets, and how to put together template sets that uniformly benefit all capabilities is unclear. Thus, techniques to find the optimal and representative template sets generated with little human effort and can be relied upon for holistic evaluation are an imminent challenge and makes quality estimation of templates and template set an important open question.

4.4 Experience of Annotators

Since annotators are an indispensable part of this study, it was important to understand their perspective. We thus interviewed the annotators in order to gain insight into their experience.

For our study, the annotators can be considered as 'experts' [5]. The common feedback we received was that it was difficult to come up with the template sets from scratch. On further probing, this difficulty could be broken down into multiple steps.

First, generating offensive content templates needs specific vocabulary, also known as lexicons in Checklist. Creating these lexicons from scratch can be subject to creativity and offensive language usage. Further, using these lexicons to generate templates is again subject to creativity, which varies from person to person, and can be difficult to replicate from a scientific perspective. This difficulty can be ported to almost any task for which templates are to be generated as it would need the creation of specific lexicon vocabulary and their combinations.

Secondly, it is not easy to ascertain what set of templates is best. As discussed in the template quality section, while the quality of individual templates can be judged by failure rates, for an annotator developing templates in a limited time-frame, the template set generated may not always be optimal, or the best possible set that finds the maximum bugs. Thus, without instant model feedback, deciding which templates are good and which are not is difficult, and finding the most optimal template set may not be feasible.

[5]educated in English and having understanding equivalent to graduate-level courses in natural language processing and machine learning

Finally, from the perspective of this particular task of the case-study, offensive content itself is a topic open to interpretation from perspectives of communities coming from varied socio-cultural backgrounds and individual sentiments, philosophy, and beliefs. What one person may find offensive, another person may not, and vice versa. As a result, despite well-documented qualitative guidelines of expectations from the models, individual examples can have debatable annotation. This ambiguity is also carried into the template generation process, where an annotator's individuality may reflect in the offensive templates that they generate.

Typically, it is easier for humans to verify annotations or explanations rather than generating them from scratch. This can be extended to judging whether a template is correct and useful. Thus developing techniques that can be utilized for automated template creation from small seed data followed by verification and labeling by humans can be an important future research direction.

4.5 Multilinguality

Given the rapid adoption of massive multilingual systems in NLP, there is an increasing need for evaluation in other languages. Thus it is intuitive to feel the need to use Checklist for multilingual models. However, template generation would typically need a native or fluent speaker of the language. It can often be difficult for researchers building massive multilingual systems to find experts fluent in multiple or specific languages. While the open-source Checklist framework provides limited capability of generating multilingual templates, it is not powerful enough to automate the process for different languages without sufficient human supervision. Thus, developing ways to create multilingual Checklists using Checklists in one language easily has immense scope.

5 Conclusion

State-of-the-art systems have been known to break down when deployed in the wild because of heavy reliance on static evaluation benchmarks that fail to holistically test the system. Several non-standard forms of evaluation into specific aspects of the models can lead to insights that might otherwise go unnoticed. Human-in-loop processes have been known to aid better explainability, trust, debugging, and improvement of NLP models by combining automation with human-expertise of language use. The Checklist framework introduced a behavioral testing approach for finding bugs in NLP models, which showed that state-of-the-art systems fail on the simplest of capabilities.

We presented a case study of using Checklist to debug an English offensive content detection system. The process we utilized was two-staged: First, we employed a human annotator to generate templates for evaluating specific model capabilities. These results were leveraged to find bugs, or capabilities in which the model is not performing as per expectation. The second step was to augment the data generated from these templates and re-train the model. This led to targeted bug-fixing and better performance not just on the test sets created from the same templates, but more generally, on independently created template sets.

Using this technique led to not only improved models but also a better understanding of the limitations and capabilities of the model in context of specific requirements. Our findings add to the growing optimism of using human expertise and non-standard evaluation to improve performance, better explainability, and increase trust in NLP systems deployed in real-world uncontrolled usage environments.

We also discuss various challenges of employing such a human-in-loop strategy. These include resource requirements, different methods to improve the model, determining the quality of templates and template sets, finding the optimal and representative template set, the difficulty for human subjects to create templates from scratch, and extension of the paradigm to languages other than English. This leads to the conclusion that the process, even though beneficial, leaves many open questions that need to be addressed.

We hope that our work further increases attention to the Checklist paradigm and motivates researchers to evaluate and improve black box NLP models using non-standard and explainable human-in-loop evaluation and investigate its challenges.

Acknowledgments

We would like to thank our colleagues Monojit Choudhary, Kalika Bali, Somak Aditya, Anirudh Srinivasan, and Karthikeyan K for their invaluable inputs during the course of this work. We extend gracious thanks to our five amazing annotators who put a lot of time and effort in understanding the problem, creating templates, and giving us their feedback.

References

Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515.

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. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1161–1166, Hong Kong, China. Association for Computational Linguistics.

Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. 2018. Hate speech dataset from a white supremacy forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium. Association for Computational Linguistics.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking NLI systems with sentences that require simple lexical inferences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 650–655, Melbourne, Australia. Association for Computational Linguistics.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6325–6334. IEEE Computer Society.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online. Association for Computational Linguistics.

Christopher Potts, Zhengxuan Wu, Atticus Geiger, and Douwe Kiela. 2020. Dynasent: A dynamic benchmark for sentiment analysis. *arXiv preprint arXiv:2012.15349*.

Vinodkumar Prabhakaran, Ben Hutchinson, and Margaret Mitchell. 2019. Perturbation sensitivity analysis to detect unintended model biases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5740–5745, Hong Kong, China. Association for Computational Linguistics.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. Do imagenet classifiers generalize to imagenet? In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5389–5400. PMLR.

Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. 2019. Are red roses red? evaluating consistency of question-answering models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6174–6184, Florence, Italy. Association for Computational Linguistics.

Marco Túlio 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, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018a. Anchors: High-precision model-agnostic explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1527–1535. AAAI Press.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018b. Semantically equivalent adversarial rules for debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, Melbourne, Australia. Association for Computational Linguistics.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Barbara Rychalska, Dominika Basaj, Alicja Gosiewska, and Przemysław Biecek. 2019. Models in the wild: On corruption robustness of neural nlp systems. In *International Conference on Neural Information Processing*, pages 235–247. Springer.

Masatoshi Tsuchiya. 2018. Performance impact caused by hidden bias of training data for recognizing textual entailment. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Appendix

Model	Characterization	Negation	Violence	Unsafe	Robustness
Base	21.23	2.59	0.2	28.3	24.17
Aug-2	0.01*	0*	0*	0.06*	0.00*
Aug-3	19.93	1.35	0.2	28.25	17.24
Aug-4	11.85	2.19	0.3	6.27	11.49
Aug-5	19.22	1.82	0.2	29.02	17.84

Table 4: Failure Rates (%) of grouped capabilities on Template Set - 2. (* refers to tested on the test holdout set such that the testing examples are disjoint from training data of the augmented model but come from the same template set)

Model	Characterization	Negation	Violence	Unsafe	Robustness
Base	21.21	18.27	20.98		35.26
Aug-2	16.88	3.90	12.43		24.22
Aug-3	0*	0*	0*	Annotator did not create Template	0.01*
Aug-4	23.56	9.15	15.69		32.04
Aug-5	15.03	7.24	17.35		22.88

Table 5: Failure Rates (%) of grouped capabilities on Template Set - 3. (* refers to tested on the test holdout set such that the testing examples are disjoint from training data of the augmented model but come from the same template set)

Model	Characterization	Negation	Violence	Unsafe	Robustness
Base	12.99	21.01	32.98	56.03	32.63
Aug-2	8.50	12.56	9.65	33.12	13.68
Aug-3	13.46	17.64	12.73	50.35	30.38
Aug-4	0*	0.01*	0*	0*	0*
Aug-5	9.43	11.89	15.28	43.5	21.99

Table 6: Failure Rates (%) of grouped capabilities on Template Set - 4. (* refers to tested on the test holdout set such that the testing examples are disjoint from training data of the augmented model but come from the same template set)

Model	Characterization	Negation	Violence	Unsafe	Robustness
Base	14.41	24.21	98.48		37.29
Aug-2	6.57	9.58	97.79		29.07
Aug-3	3.85	3.70	96.98	Annotator did not create Template	25.07
Aug-4	8.32	16.01	72.22		33.35
Aug-5	0*	0*	0*		0.01*

Table 7: Failure Rates (%) of grouped capabilities on Template Set - 5. (* refers to tested on the test holdout set such that the testing examples are disjoint from training data of the augmented model but come from the same template set)