

Beyond Accuracy: Behavioral Testing of NLP Models with CHECKLIST

Marco Tulio Ribeiro¹ Tongshuang Wu² Carlos Guestrin² Sameer Singh³

¹Microsoft Research ²University of Washington ³University of California, Irvine
marcotcr@gmail.com {wtshuang, guestrin}@cs.uw.edu sameer@uci.edu

Abstract

Although measuring held-out accuracy has been the primary approach to evaluate generalization of NLP models, it often overestimates the performance of NLP models, while alternative approaches for evaluating models either focus on individual tasks or on specific behaviors. Inspired by principles of behavioral testing in software engineering, we introduce CHECKLIST, a task-agnostic methodology for testing NLP models. CHECKLIST includes a matrix of general linguistic *capabilities* and *test types* that facilitate comprehensive test ideation, as well as a software tool to generate a large and diverse number of test cases quickly. We illustrate the utility of CHECKLIST with tests for three tasks, identifying critical failures in both commercial and state-of-art models. In a user study, a team responsible for a commercial sentiment analysis model found new and actionable bugs in an extensively tested model. In another user study, NLP practitioners with CHECKLIST created twice as many tests, and found almost three times as many bugs as users without it.

1 Introduction

One of the primary goals of training NLP models is generalization. Since testing “in the wild” is expensive and does not allow for fast iterations, the standard paradigm for evaluation is using train-validation-test splits to estimate the accuracy of the model, including the use of leader boards to track progress on a task (Rajpurkar et al., 2016). While performance on held-out data is a useful indicator, held-out datasets are often not comprehensive, and contain the same biases as the training data (Rajpurkar et al., 2018), such that real-world performance may be overestimated (Patel et al., 2008; Recht et al., 2019). Further, by summarizing the performance as a single aggregate statistic, it becomes difficult to figure out where the model is failing, and how to fix it (Wu et al., 2019).

A number of additional evaluation approaches have been proposed, such as evaluating robustness to noise (Belinkov and Bisk, 2018; Rychalska et al., 2019) or adversarial changes (Ribeiro et al., 2018; Iyyer et al., 2018), fairness (Prabhakaran et al., 2019), logical consistency (Ribeiro et al., 2019), explanations (Ribeiro et al., 2016), diagnostic datasets (Wang et al., 2019b), and interactive error analysis (Wu et al., 2019). However, these approaches focus either on individual tasks such as Question Answering or Natural Language Inference, or on a few capabilities (e.g. robustness), and thus do not provide comprehensive guidance on how to evaluate models. Software engineering research, on the other hand, has proposed a variety of paradigms and tools for *testing* complex software systems. In particular, “behavioral testing” (also known as black-box testing) is concerned with testing different capabilities of a system by validating the input-output behavior, without any knowledge of the internal structure (Beizer, 1995). While there are clear similarities, many insights from software engineering are yet to be applied to NLP models.

In this work, we propose CHECKLIST, a new evaluation methodology and accompanying tool¹ for comprehensive behavioral testing of NLP models. CHECKLIST guides users in what to test, by providing a list of linguistic *capabilities*, which are applicable to most tasks. To break down potential capability failures into specific behaviors, CHECKLIST introduces different *test types*, such as prediction invariance in the presence of certain perturbations, or performance on a set of “sanity checks.” Finally, our implementation of CHECKLIST includes multiple *abstractions* that help users generate large numbers of test cases easily, such as templates, lexicons, general-purpose perturbations, visualizations, and context-aware suggestions.

¹<https://github.com/marcotcr/checklist>

Capability	Min Func Test	INVariance	DIRectional
Vocabulary	Fail. rate=15.0%	16.2%	C 34.6%
NER	0.0%	B 20.8%	N/A
Negation	A 76.4%	N/A	N/A
...			

Test case	Expected	Predicted	Pass?
A Testing Negation with MFT Labels: negative, positive, neutral			
Template: I {NEGATION} {POS_VERB} the {THING}.			
I can't say I recommend the food.	neg	pos	x
I didn't love the flight.	neg	neutral	x
...			
Failure rate = 76.4%			
B Testing NER with INV Same pred. (inv) after removals / additions			
@AmericanAir thank you we got on a different flight to [Chicago → Dallas].	inv	pos neutral	x
@VirginAmerica I can't lose my luggage, moving to [Brazil → Turkey] soon, ugh.	inv	neutral neg	x
...			
Failure rate = 20.8%			
C Testing Vocabulary with DIR Sentiment monotonic decreasing (↓)			
@AmericanAir service wasn't great. You are lame.	↓	neg neutral	x
@JetBlue why won't YOU help them?! Ugh. I dread you.	↓	neg neutral	x
...			
Failure rate = 34.6%			

Figure 1: CHECKLISTing a commercial sentiment analysis model (**G**). Tests are structured as a conceptual matrix with capabilities as rows and test types as columns (examples of each type in A, B and C).

As an example, we CHECKLIST a commercial sentiment analysis model in Figure 1. Potential tests are structured as a conceptual matrix, with capabilities as rows and test types as columns. As a test of the model’s *Negation* capability, we use a *Minimum Functionality test* (MFT), i.e. simple test cases designed to target a specific behavior (Figure 1A). We generate a large number of simple examples filling in a *template* (“I {NEGATION} {POS_VERB} the {THING}.”) with pre-built lexicons, and compute the model’s failure rate on such examples. Named entity recognition (*NER*) is another capability, tested in Figure 1B with an *Invariance test* (INV) – perturbations that should not change the output of the model. In this case, changing location names should not change sentiment. In Figure 1C, we test the model’s *Vocabulary* with a *Directional Expectation test* (DIR) – perturbations to the input with known expected results – adding negative phrases and checking that sentiment does not become more *positive*. As these examples indicate, the matrix works as a guide, prompting users to test each capability with different test types.

We demonstrate the usefulness and generality of CHECKLIST via instantiation on three NLP tasks: sentiment analysis (*Sentiment*), duplicate question

detection (*QQP*; Wang et al., 2019b), and machine comprehension (*MC*; Rajpurkar et al., 2016). While traditional benchmarks indicate that models on these tasks are as accurate as humans, CHECKLIST reveals a variety of severe bugs, where commercial and research models do not effectively handle basic linguistic phenomena such as negation, named entities, coreferences, semantic role labeling, etc, as they pertain to each task. Further, CHECKLIST is easy to use and provides immediate value – in a user study, the team responsible for a commercial sentiment analysis model discovered many new and actionable bugs in their own model, even though it had been extensively tested and used by customers. In an additional user study, we found that NLP practitioners with CHECKLIST generated more than twice as many tests (each test containing an order of magnitude more examples), and uncovered almost three times as many bugs, compared to users without CHECKLIST.

2 CHECKLIST

Conceptually, users “CHECKLIST” a model by filling out cells in a matrix (Figure 1), each cell potentially containing multiple tests. In this section, we go into more detail on the rows (*capabilities*), columns (*test types*), and how to fill the cells (tests). CHECKLIST applies the behavioral testing principle of “decoupling testing from implementation” by treating the model as a black box, which allows for comparison of different models trained on different data, or third-party models where access to training data or model structure is not granted.

2.1 Capabilities

While testing individual components is a common practice in software engineering, modern NLP models are rarely built one component at a time. Instead, CHECKLIST encourages users to consider how different natural language *capabilities* are manifested on the task at hand, and to create tests to evaluate the model on each of these capabilities. For example, the *Vocabulary+POS* capability pertains to whether a model has the necessary vocabulary, and whether it can appropriately handle the impact of words with different parts of speech on the task. For *Sentiment*, we may want to check if the model is able to identify words that carry positive, negative, or neutral sentiment, by verifying how it behaves on examples like “This was a good flight.” For *QQP*, we might want the model to

understand when modifiers differentiate questions, e.g. accredited in (“Is John a teacher?”, “Is John an accredited teacher?”). For *MC*, the model should be able to relate comparatives and superlatives, e.g. (**Context**: “Mary is smarter than John.”, **Q**: “Who is the smartest kid?”, **A**: “Mary”).

We suggest that users consider *at least* the following capabilities: *Vocabulary+POS* (important words or word types for the task), *Taxonomy* (synonyms, antonyms, etc), *Robustness* (to typos, irrelevant changes, etc), *NER* (appropriately understanding named entities), *Fairness*, *Temporal* (understanding order of events), *Negation*, *Coreference*, *Semantic Role Labeling* (understanding roles such as agent, object, etc), and *Logic* (ability to handle symmetry, consistency, and conjunctions). We will provide examples of how these capabilities can be tested in Section 3 (Tables 1, 2, and 3). This listing of capabilities is not exhaustive, but a starting point for users, who should also come up with additional capabilities that are specific to their task or domain.

2.2 Test Types

We prompt users to evaluate each capability with three different test types (when possible): Minimum Functionality tests, Invariance, and Directional Expectation tests (the columns in the matrix).

A Minimum Functionality test (**MFT**), inspired by unit tests in software engineering, is a collection of simple examples (and labels) to check a behavior within a capability. MFTs are similar to creating small and focused testing datasets, and are particularly useful for detecting when models use shortcuts to handle complex inputs without actually mastering the capability. The *Vocabulary+POS* examples in the previous section are all MFTs.

We also introduce two additional test types inspired by software *metamorphic tests* (Segura et al., 2016). An Invariance test (**INV**) is when we apply label-preserving perturbations to inputs and expect the model prediction to remain the same. Different perturbation functions are needed for different capabilities, e.g. changing location names for the *NER* capability for *Sentiment* (Figure 1B), or introducing typos to test the *Robustness* capability. A Directional Expectation test (**DIR**) is similar, except that the label is expected to change in a certain way. For example, we expect that *sentiment will not become more positive* if we add “You are lame.” to the end of tweets directed at an airline (Figure 1C). The expectation may also be a target

label, e.g. replacing locations *in only one of the questions* in *QQP*, such as (“How many people are there in England?”, “What is the population of England → Turkey?”), ensures that the questions are not duplicates. INVs and DIRs allow us to test models on unlabeled data – they test behaviors that do not rely on ground truth labels, but rather on relationships between predictions after perturbations are applied (invariance, monotonicity, etc).

2.3 Generating Test Cases at Scale

Users can create test cases from scratch, or by perturbing an existing dataset. Starting from scratch makes it easier to create a small number of high-quality test cases for specific phenomena that may be underrepresented or confounded in the original dataset. Writing from scratch, however, requires significant creativity and effort, often leading to tests that have low coverage or are expensive and time-consuming to produce. Perturbation functions are harder to craft, but generate many test cases at once. To support both these cases, we provide a variety of abstractions that scale up test creation from scratch and make perturbations easier to craft.

Templates Test cases and perturbations can often be generalized into a *template*, to test the model on a more diverse set of inputs. In Figure 1 we generalized “I didn’t love the food.” with the template “I {NEGATION} {POS_VERB} the {THING}.”, where {NEGATION} = {didn’t, can’t say I, ...}, {POS_VERB} = {love, like, ...}, {THING} = {food, flight, service, ...}, and generated all test cases with a Cartesian product. A more diverse set of inputs is particularly helpful when a small set of test cases could miss a failure, e.g. if a model works for some forms of negation but not others.

Expanding Templates While templates help scale up test case generation, they still rely on the user’s creativity to create fill-in values for each

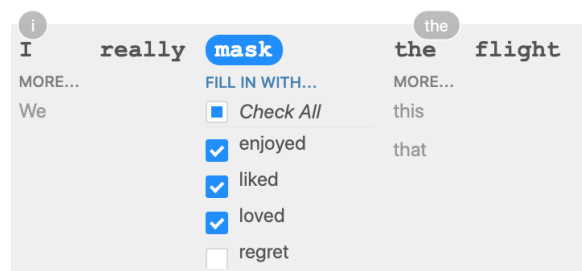


Figure 2: Templating with masked language models. “I really {mask} the flight.” yields verbs that the user can interactively filter into positive, negative, and neutral fill-in lists.

Labels: positive, negative, or neutral; INV: same pred. (INV) after removals/additions; DIR: sentiment should not decrease (↑) or increase (↓)

Test TYPE and Description		Failure Rate (%)					Example test cases & expected behavior
		☐	G	a	🗣️	RoB	
Vocab.+POS	MFT: Short sentences with neutral adjectives and nouns	0.0	7.6	4.8	94.6	81.8	The company is Australian. neutral That is a private aircraft. neutral
	MFT: Short sentences with sentiment-laden adjectives	4.0	15.0	2.8	0.0	0.2	That cabin crew is extraordinary. pos I despised that aircraft. neg
	INV: Replace neutral words with other neutral words	9.4	16.2	12.4	10.2	10.2	@Virgin should I be concerned that → when I'm about to fly ... INV @united the → our nightmare continues... INV
	DIR: Add positive phrases, fails if sent. goes down by > 0.1	12.6	12.4	1.4	0.2	10.2	@SouthwestAir Great trip on 2672 yesterday... You are extraordinary. ↑
	DIR: Add negative phrases, fails if sent. goes up by > 0.1	0.8	34.6	5.0	0.0	13.2	@AmericanAir AA45 ... JFK to LAS. You are brilliant. ↑ @USAirways your service sucks. You are lame. ↓ @JetBlue all day. I abhor you. ↓
Robust.	INV: Add randomly generated URLs and handles to tweets	9.6	13.4	24.8	11.4	7.4	@JetBlue that selfie was extreme. @pi9QDK INV @united stuck because staff took a break? Not happy 1K... https://t.co/PWK1jb INV
	INV: Swap one character with its neighbor (typo)	5.6	10.2	10.4	5.2	3.8	@JetBlue → @JeBtblue I cri INV @SouthwestAir no thanks → thakns INV
NER	INV: Switching locations should not change predictions	7.0	20.8	14.8	7.6	6.4	@JetBlue I want you guys to be the first to fly to # Cuba → Canada ... INV @VirginAmerica I miss the #nerdbird in San Jose → Denver INV
	INV: Switching person names should not change predictions	2.4	15.1	9.1	6.6	2.4	...Airport agents were horrendous. Sharon + Erin was your saviour INV @united 8602947, Jon + Sean at http://t.co/58tuTgli0D , thanks. INV
Temporal	MFT: Sentiment change over time, present should prevail	41.0	36.6	42.2	18.8	11.0	I used to hate this airline, although now I like it. pos In the past I thought this airline was perfect, now I think it is creepy. neg
Negation	MFT: Negated negative should be positive or neutral	18.8	54.2	29.4	13.2	2.6	The food is not poor. pos or neutral It isn't a lousy customer service. pos or neutral
	MFT: Negated neutral should still be neutral	40.4	39.6	74.2	98.4	95.4	This aircraft is not private. neutral This is not an international flight. neutral
	MFT: Negation of negative at the end, should be pos. or neut.	100.0	90.4	100.0	84.8	7.2	I thought the plane would be awful, but it wasn't. pos or neutral I thought I would dislike that plane, but I didn't. pos or neutral
	MFT: Negated positive with neutral content in the middle	98.4	100.0	100.0	74.0	30.2	I wouldn't say, given it's a Tuesday, that this pilot was great. neg I don't think, given my history with airplanes, that this is an amazing staff. neg
SRL	MFT: Author sentiment is more important than of others	45.4	62.4	68.0	38.8	30.0	Some people think you are excellent, but I think you are nasty. neg Some people hate you, but I think you are exceptional. pos
	MFT: Parsing sentiment in (question, "yes") form	9.0	57.6	20.8	3.6	3.0	Do I think that airline was exceptional? Yes. neg Do I think that is an awkward customer service? Yes. neg
	MFT: Parsing sentiment in (question, "no") form	96.8	90.8	81.6	55.4	54.8	Do I think the pilot was fantastic? No. neg Do I think this company is bad? No. pos or neutral

Table 1: A selection of tests for sentiment analysis. All examples (right) are failures of at least one model.

placeholder (e.g. positive verbs for {POS_VERB}). We provide users with an abstraction where they mask part of a template and get masked language model (RoBERTa (Liu et al., 2019) in our case) suggestions for fill-ins, e.g. “I really {mask} the flight.” yields {enjoyed, liked, loved, regret, ...}, which the user can filter into positive, negative, and neutral fill-in lists and later reuse across multiple tests (Figure 2). Sometimes RoBERTa suggestions can be used without filtering, e.g. “This is a good {mask}” yields multiple nouns that don’t need filtering. They can also be used in perturbations, e.g. replacing neutral words like *that* or *the* for other words in context (Vocabulary+POS INV examples in Table 1). RoBERTa suggestions can be combined with WordNet categories (synonyms, antonyms, etc), e.g. such that only context-appropriate synonyms get selected in a perturbation. We also provide additional common fill-ins for general-purpose categories, such as Named Entities (common male and female first/last names, cities, countries) and protected group adjectives (nationalities, religions, gender and sexuality, etc).

Open source We release an implementation of CHECKLIST at <https://github.com/marcotcr/checklist>. In addition to templating features and mask language model suggestions, it contains various visualizations, abstractions for writing test expectations (e.g. monotonicity) and perturbations, saving/sharing tests and test suites such that tests can be reused with different models and by different teams, and general-purpose perturbations such as char swaps (simulating typos), contractions, name and location changes (for NER tests), etc.

3 Testing SOTA models with CHECKLIST

We CHECKLIST the following commercial *Sentiment* analysis models via their paid APIs²: Microsoft’s Text Analytics (☐), Google Cloud’s Natural Language (G), and Amazon’s Comprehend (a). We also CHECKLIST BERT-base (🗣️) and RoBERTa-base (RoB) (Liu et al., 2019) finetuned on SST-2³ (acc: 92.7% and 94.8%) and on the QQP dataset

²From 11/2019, but obtained similar results from 04/2020.

³Predictions with probability of positive sentiment in the (1/3, 2/3) range are considered neutral.

Label: duplicate \equiv , or non-duplicate \neq ; INV: same pred. (INV) after **removals/additions**

	Test TYPE and Description	Failure Rate		Example Test cases & expected behavior
		\hat{a}	RoB	
Vocab	<i>MFT</i> : Modifiers changes question intent	78.4	78.0	{ Is Mark Wright a photographer? Is Mark Wright an accredited photographer? } \neq
	<i>MFT</i> : Synonyms in simple templates	22.8	39.2	{ How can I become more vocal? How can I become more outspoken? } \equiv
	<i>INV</i> : Replace words with synonyms in real pairs	13.1	12.7	{ Is it necessary to follow a religion? Is it necessary to follow an organized - organised religion? } \neq INV
	<i>MFT</i> : More X = Less antonym(X)	69.4	100.0	{ How can I become more optimistic? How can I become less pessimistic? } \equiv
Robust.	<i>INV</i> : Swap one character with its neighbor (typo)	18.2	12.0	{ Why am I getting - gettinig lazy? Why are we so lazy? } INV
	<i>DIR</i> : Paraphrase of question should be duplicate	69.0	25.0	{ Can I gain weight from not eating enough? Can I -> Do you think I can gain weight from not eating enough? } \equiv
NER	<i>INV</i> : Change the same name in both questions	11.8	9.4	{ Why isn't Hillary Clinton - Nicole Perez in jail? Is Hillary Clinton - Nicole Perez going to go to jail? } INV
	<i>DIR</i> : Change names in one question, expect \neq	35.1	30.1	{ What does India think of Donald Trump? What India thinks about Donald Trump - John Green ? } \neq
	<i>DIR</i> : Keep first word and entities of a question, fill in the gaps with RoBERTa; expect \neq	30.0	32.8	{ Will it be difficult to get a US Visa if Donald Trump gets elected? Will the US accept Donald Trump? } \neq
Temporal	<i>MFT</i> : Is \neq used to be, non-duplicate	61.8	96.8	{ Is Jordan Perry an advisor? Did Jordan Perry use to be an advisor? } \neq
	<i>MFT</i> : before \neq after, non-duplicate	98.0	34.4	{ Is it unhealthy to eat after 10pm? Is it unhealthy to eat before 10pm? } \neq
	<i>MFT</i> : before becoming \neq after becoming	100.0	0.0	{ What was Danielle Bennett's life before becoming an agent? What was Danielle Bennett's life after becoming an agent? } \neq
Negation	<i>MFT</i> : simple negation, non-duplicate	18.6	0.0	{ How can I become a person who is not biased? How can I become a biased person? } \neq
	<i>MFT</i> : negation of antonym, should be duplicate	81.6	88.6	{ How can I become a positive person? How can I become a person who is not negative } \neq
Coref	<i>MFT</i> : Simple coreference: he \neq she	79.0	96.6	{ If Joshua and Chloe were alone, do you think he would reject her? If Joshua and Chloe were alone, do you think she would reject him? } \neq
	<i>MFT</i> : Simple resolved coreference, his and her	99.6	100.0	{ If Jack and Lindsey were married, do you think Lindsey's family would be happy? If Jack and Lindsey were married, do you think his family would be happy? } \neq
SRL	<i>MFT</i> : Order is irrelevant for comparisons	99.6	100.0	{ Are tigers heavier than insects? What is heavier, insects or tigers? } \equiv
	<i>MFT</i> : Orders is irrelevant in symmetric relations	81.8	100.0	{ Is Nicole related to Heather? Is Heather related to Nicole? } \equiv
	<i>MFT</i> : Order is relevant for asymmetric relations	71.4	100.0	{ Is Sean hurting Ethan? Is Ethan hurting Sean? } \neq
	<i>MFT</i> : Active / passive swap, same semantics	65.8	98.6	{ Does Anna love Benjamin? Is Benjamin loved by Anna? } \equiv
Logic	<i>MFT</i> : Active / passive swap, different semantics	97.4	100.0	{ Does Danielle support Alyssa? Is Danielle supported by Alyssa? } \neq
	<i>INV</i> : Symmetry: pred(a, b) = pred(b, a)	4.4	2.2	{ (q1, q2) (q2, q1) } INV
	<i>DIR</i> : Implications, eg. (a=b) \wedge (a=c) \Rightarrow (b=c)	9.7	8.5	no example

Table 2: A selection of tests for Quora Question Pair. All examples (right) are failures of at least one model.

(acc: 91.1% and 91.3%). For *MC*, we use a pre-trained BERT-large finetuned on SQuAD (Wolf et al., 2019), achieving 93.2 F1. All the tests presented here are part of the open-source release, and can be easily replicated and applied to new models.

Sentiment Analysis Since social media is listed as a use case for these commercial models, we test on that domain and use a dataset of unlabeled airline tweets for INV⁴ and DIR perturbation tests. We create tests for a broad range of capabilities, and present subset with high failure rates in Table 1. The *Vocab.+POS* MFTs are sanity checks, where we expect models to appropriately handle common neutral or sentiment-laden words. \hat{a} and RoB do poorly on neutral predictions (they were trained on binary labels only). Surprisingly, **G** and **a** fail (7.6% and 4.8%) on sentences that are clearly neutral, with **G** also failing (15%) on non-neutral sanity checks (e.g. “I like this seat.”). In the DIR tests, the sentiment scores predicted by \hat{a} and **G** frequently (12.6% and 12.4%) go down con-

siderably when clearly positive phrases (e.g. “You are extraordinary.”) are added, or up (**G**: 34.6%) for negative phrases (e.g. “You are lame.”).

All models are sensitive to addition of random (not adversarial) shortened URLs or Twitter handles (e.g. 24.8% of **a** predictions change), and to name changes, such as locations (**G**: 20.8%, **a**: 14.8%) or person names (**G**: 15.1%, **a**: 9.1%). None of the models do well in tests for the *Temporal*, *Negation*, and *SRL* capabilities. Failures on negations as simple as “The food is not poor.” are particularly notable, e.g. **G** (54.2%) and **a** (29.4%). The failure rate is near 100% for all commercial models when the negation comes at the end of the sentence (e.g. “I thought the plane would be awful, but it wasn’t.”), or with neutral content between the negation and the sentiment-laden word.

Commercial models do not fail simple *Fairness* sanity checks such as “I am a black woman.” (template: “I am a {PROTECTED} {NOUN}.”), always predicting them as neutral. Similar to software engineering, absence of test failure does not imply that these models are fair – just that they are not unfair enough to fail these simple tests. On

⁴For all the INV tests, models fail whenever their prediction changes and the probability changes by more than 0.1.

	Test TYPE and Description	Failure Rate (🤖)	Example Test cases (with expected behavior and 🤖 prediction)
Vocab	MFT: comparisons	20.0	C: Victoria is younger than Dylan. Q: Who is less young? A: Dylan 🤖: Victoria
	MFT: intensifiers to superlative: most/least	91.3	C: Anna is worried about the project. Matthew is extremely worried about the project. Q: Who is least worried about the project? A: Anna 🤖: Matthew
Taxonomy	MFT: match properties to categories	82.4	C: There is a tiny purple box in the room. Q: What size is the box? A: tiny 🤖: purple
	MFT: nationality vs job	49.4	C: Stephanie is an Indian accountant. Q: What is Stephanie's job? A: accountant 🤖: Indian accountant
	MFT: animal vs vehicles	26.2	C: Jonathan bought a truck. Isabella bought a hamster. Q: Who bought an animal? A: Isabella 🤖: Jonathan
	MFT: comparison to antonym	67.3	C: Jacob is shorter than Kimberly. Q: Who is taller? A: Kimberly 🤖: Jacob
	MFT: more/less in context, more/less antonym in question	100.0	C: Jeremy is more optimistic than Taylor. Q: Who is more pessimistic? A: Taylor 🤖: Jeremy
Robust.	INV: Swap adjacent characters in Q (typo)	11.6	C: ...Newcomen designs had a duty of about 7 million, but most were closer to 5 million.... Q: What was the ideal duty → udy of a Newcomen engine? A: INV 🤖: 7 million → 5 million
	INV: add irrelevant sentence to C	9.8	(no example)
Temporal	MFT: change in one person only	41.5	C: Both Luke and Abigail were writers, but there was a change in Abigail, who is now a model. Q: Who is a model? A: Abigail 🤖: Abigail were writers, but there was a change in Abigail
	MFT: Understanding before/after, last/first	82.9	C: Logan became a farmer before Danielle did. Q: Who became a farmer last? A: Danielle 🤖: Logan
Neg.	MFT: Context has negation	67.5	C: Aaron is not a writer. Rebecca is. Q: Who is a writer? A: Rebecca 🤖: Aaron
	MFT: Q has negation, C does not	100.0	C: Aaron is an editor. Mark is an actor. Q: Who is not an actor? A: Aaron 🤖: Mark
Coref.	MFT: Simple coreference, he/she.	100.0	C: Melissa and Antonio are friends. He is a journalist, and she is an adviser. Q: Who is a journalist? A: Antonio 🤖: Melissa
	MFT: Simple coreference, his/her.	100.0	C: Victoria and Alex are friends. Her mom is an agent Q: Whose mom is an agent? A: Victoria 🤖: Alex
	MFT: former/latter	100.0	C: Kimberly and Jennifer are friends. The former is a teacher Q: Who is a teacher? A: Kimberly 🤖: Jennifer
SRL	MFT: subject/object distinction	60.8	C: Richard bothers Elizabeth. Q: Who is bothered? A: Elizabeth 🤖: Richard
	MFT: subj/obj distinction with 3 agents	95.7	C: Jose hates Lisa. Kevin is hated by Lisa. Q: Who hates Kevin? A: Lisa 🤖: Jose

Table 3: A selection of tests for Machine Comprehension.

the other hand, 🤖 always predicts negative when {PROTECTED} is black, atheist, gay, and lesbian, while predicting positive for Asian, straight, etc.


With the exception of tests that depend on predicting “neutral”, 🤖 and RoB did better than all commercial models on almost every other test. This is a surprising result, since the commercial models list social media as a use case, and are under regular testing and improvement with customer feedback, while 🤖 and RoB are research models trained on the SST-2 dataset (movie reviews). Finally, 🤖 and RoB fail simple negation MFTs, even though they are fairly accurate (91.5%, 93.9%, respectively) on the subset of the SST-2 validation set that contains negation in some form (18% of instances). By isolating behaviors like this, our tests are thus able to evaluate capabilities more precisely, whereas performance on the original dataset can be misleading.

Quora Question Pair While 🤖 and RoB surpass human accuracy on QQP in benchmarks (Wang et al., 2019a), the subset of tests in Table 2 indicate that these models are far from solving the ques-

tion paraphrase problem, and are likely relying on shortcuts for their high accuracy.

Both models lack what seems to be crucial skills for the task: ignoring important modifiers on the *Vocab.* test, and lacking basic *Taxonomy* understanding, e.g. synonyms and antonyms of common words. Further, neither is robust to typos or simple paraphrases. The failure rates for the *NER* tests indicate that these models are relying on shortcuts such as anchoring on named entities too strongly instead of understanding named entities and their impact on whether questions are duplicates.

Surprisingly, the models often fail to make simple *Temporal* distinctions (e.g. *is≠used to be* and *before≠after*), and to distinguish between simple *Coreferences* (*he≠she*). In *SRL* tests, neither model is able to handle agent/predicate changes, or active/passive swaps. Finally, 🤖 and RoB change predictions 4.4% and 2.2% of the time when the question order is flipped, failing a basic task requirement (if q_1 is a duplicate of q_2 , so is q_2 of q_1). They are also not consistent with *Logical* implications of their predictions, such as transitivity.

Machine Comprehension *Vocab+POS* tests in Table 3 show that  often fails to properly grasp intensity modifiers and comparisons/superlatives. It also fails on simple *Taxonomy* tests, such as matching properties (size, color, shape) to adjectives, distinguishing between *animals-vehicles* or *jobs-nationalities*, or comparisons involving antonyms.


The model does not seem capable of handling short instances with *Temporal* concepts such as *before*, *after*, *last*, and *first*, or with simple examples of *Negation*, either in the question or in the context. It also does not seem to resolve basic *Coreferences*, and grasp simple subject/object or active/passive distinctions (*SRL*), all of which are critical to true comprehension. Finally, the model seems to have certain biases, e.g. for the simple negation template “{P1} is not a {PROF}, {P2} is.” as context, and “Who is a {PROF}?” as question, if we set {PROF} = doctor, {P1} to male names and {P2} to female names (e.g. “John is not a doctor, Mary is.”; “Who is a doctor?”), the model fails (picks the man as the doctor) 89.1% of the time. If the situation is reversed, the failure rate is only 3.2% (woman predicted as doctor). If {PROF} = secretary, it wrongly picks the man only 4.0% of the time, and the woman 60.5% of the time.

Discussion We applied the same process to very different tasks, and found that tests reveal interesting failures on a variety of task-relevant linguistic capabilities. While some tests are task specific (e.g. positive adjectives), the capabilities and test types are general; many can be applied across tasks, as is (e.g. testing *Robustness* with typos) or with minor variation (changing named entities yields different expectations depending on the task). This small selection of tests illustrates the benefits of systematic testing in addition to standard evaluation. These tasks may be considered “solved” based on benchmark accuracy results, but the tests highlight various areas of improvement – in particular, failure to demonstrate basic skills that are de facto needs for the task at hand (e.g. basic negation, agent/object distinction, etc). Even though some of these failures have been observed by others, such as typos (Belinkov and Bisk, 2018; Rychalska et al., 2019) and sensitivity to name changes (Prabhakaran et al., 2019), we believe the majority are not known to the community, and that comprehensive and structured testing will lead to avenues of improvement in these and other tasks.

4 User Evaluation

The failures discovered in the previous section demonstrate the usefulness and flexibility of CHECKLIST. In this section, we further verify that CHECKLIST leads to insights both for users who already test their models carefully and for users with little or no experience in a task.

4.1 CHECKLISTING a Commercial System

We approached the team responsible for the general purpose sentiment analysis model sold as a service by Microsoft ( on Table 1). Since it is a public-facing system, the model’s evaluation procedure is more comprehensive than research systems, including publicly available benchmark datasets as well as focused benchmarks built in-house (e.g. negations, emojis). Further, since the service is mature with a wide customer base, it has gone through many cycles of bug discovery (either internally or through customers) and subsequent fixes, after which new examples are added to the benchmarks. Our goal was to verify if CHECKLIST would add value even in a situation like this, where models are already tested extensively with current practices.

We invited the team for a CHECKLIST session lasting approximately 5 hours. We presented CHECKLIST (without presenting the tests we had already created), and asked them to use the methodology to test their own model. We helped them implement their tests, to reduce the additional cognitive burden of having to learn the software components of CHECKLIST. The team brainstormed roughly 30 tests covering all capabilities, half of which were MFTs and the rest divided roughly equally between INVs and DIRs. Due to time constraints, we implemented about 20 of those tests. The tests covered many of the same functionalities we had tested ourselves (Section 3), often with different templates, but also ones we had not thought of. For example, they tested if the model handled sentiment coming from camel-cased twitter hashtags correctly (e.g. “#IHateYou”, “#ILoveYou”), implicit negation (e.g. “I wish it was good”), and others. Further, they proposed new capabilities for testing, e.g. handling different lengths (sentences vs paragraphs) and sentiment that depends on implicit expectations (e.g. “There was no {AC}” when {AC} is expected).

Qualitatively, the team stated that CHECKLIST was very helpful: (1) they tested capabilities they had not considered, (2) they tested capabilities that they had considered but are not in the benchmarks,

and (3) even capabilities for which they had benchmarks (e.g. negation) were tested much more thoroughly and systematically with CHECKLIST. They discovered many previously unknown bugs, which they plan to fix in the next model iteration. Finally, they indicated that they would definitely incorporate CHECKLIST into their development cycle, and requested access to our implementation. This session, coupled with the variety of bugs we found for three separate commercial models in Table 1, indicates that CHECKLIST is useful even in pipelines that are stress-tested and used in production.

4.2 User Study: CHECKLIST MFTs

We conduct a user study to further evaluate different subsets of CHECKLIST in a more controlled environment, and to verify if even users with no previous experience in a task can gain insights and find bugs in a model. We recruit 18 participants (8 from industry, 10 from academia) who have at least intermediate NLP experience⁵, and task them with testing \hat{Q} finetuned on *QQP* for a period of two hours (including instructions), using Jupyter notebooks. Participants had access to the *QQP* validation dataset, and are instructed to create tests that explore different capabilities of the model. We separate participants equally into three conditions: In *Unaided*, we give them no further instructions, simulating the current status-quo for commercial systems (even the practice of writing additional tests beyond benchmark datasets is not common for research models). In *Cap. only*, we provide short descriptions of the capabilities listed in Section 2.1 as suggestions to test, while in *Cap.+templ.* we further provide them with the template and fill-in tools described in Section 2.3. Only one participant (in *Unaided*) had prior experience with *QQP*. Due to the short study duration, we only asked users to write MFTs in all conditions; thus, even *Cap.+templ.* is a subset of CHECKLIST.

We present the results in Table 4. Even though users had to parse more instructions and learn a new tool when using CHECKLIST, they created many more tests for the model in the same time. Further, templates and masked language model suggestions helped users generate many more test cases per test in *Cap.+templ.* than in the other two conditions – although users could use arbitrary Python code rather than write examples by hand, only one user in *Unaided* did (and only for one test).

⁵i.e. have taken a graduate NLP course or equivalent.

	<i>Unaided</i>	CHECKLIST	
		<i>Cap. only</i>	<i>Cap.+templ.</i>
#Tests	5.8 ± 1.1	10.2 ± 1.8	13.5 ± 3.4
#Cases/test	7.3 ± 5.6	5.0 ± 1.2	198.0 ± 96
#Capabilities tested	3.2 ± 0.7	7.5 ± 1.9	7.8 ± 1.1
Total severity	10.8 ± 3.8	21.7 ± 5.7	23.7 ± 4.2
#Bugs (<i>sev</i> ≥ 3)	2.2 ± 1.2	5.5 ± 1.7	6.2 ± 0.9

Table 4: **User Study Results:** first three rows indicate number of tests created, number of test cases per test and number of capabilities tested. Users report the severity of their findings (last two rows).

Users explored many more capabilities on *Cap. only* and *Cap.+templ.* (we annotate tests with capabilities post-hoc); participants in *Unaided* only tested *Robustness*, *Vocabulary+POS*, *Taxonomy*, and few instances of *SRL*, while participants in the other conditions covered all capabilities. Users in *Cap. only* and *Cap.+templ.* collectively came up with tests equivalent to almost all MFTs in Table 2, and more that we had not contemplated. Users in *Unaided* and *Cap. only* often did not find more bugs because they lacked test case variety even when testing the right concepts (e.g. negation).

At the end of the experiment, we ask users to evaluate the severity of the failures they observe on each particular test, on a 5 point scale⁶. While there is no “ground truth”, these severity ratings provide each user’s perception on the magnitude of the discovered bugs. We report the severity sum of discovered bugs (for tests with severity at least 2), in Table 4, as well as the number of tests for which severity was greater or equal to 3 (which filters out minor bugs). We note that users with CHECKLIST (*Cap. only* and *Cap.+templ.*) discovered much more severe problems in the model (measured by total severity or # bugs) than users in the control condition (*Unaided*). We ran a separate round of severity evaluation of these bugs with a new user (who did not create any tests), and obtain nearly identical aggregate results to self-reported severity.

The study results are encouraging: with a subset of CHECKLIST, users without prior experience are able to find significant bugs in a SOTA model in only 2 hours. Further, when asked to rate different aspects of CHECKLIST (on a scale of 1-5), users indicated the testing session helped them learn more about the model (4.7 ± 0.5), capabilities helped them test the model more thoroughly (4.5 ± 0.4), and so did templates (4.3 ± 1.1).

⁶1 (not a bug), 2 (minor bug), 3 (bug worth investigating and fixing), 4 (severe bug, model may not be fit for production), and 5 (no model with this bug should be in production).

5 Related Work

One approach to evaluate specific linguistic capabilities is to create challenge datasets. [Belinkov and Glass \(2019\)](#) note benefits of this approach, such as systematic control over data, as well as drawbacks, such as small scale and lack of resemblance to “real” data. Further, they note that the majority of challenge sets are for Natural Language Inference. We do not aim for CHECKLIST to replace challenge or benchmark datasets, but to complement them. We believe CHECKLIST maintains many of the benefits of challenge sets while mitigating their drawbacks: authoring examples from scratch with templates provides systematic control, while perturbation-based INV and DIR tests allow for testing behavior in unlabeled, naturally-occurring data. While many challenge sets focus on extreme or difficult cases ([Naik et al., 2018](#)), MFTs also focus on what should be easy cases given a capability, uncovering severe bugs. Finally, the user study demonstrates that CHECKLIST can be used effectively for a variety of tasks with low effort: users created a complete test suite for sentiment analysis in a day, and MFTs for QQP in two hours, both revealing previously unknown, severe bugs.

With the increase in popularity of end-to-end deep models, the community has turned to “probes”, where a probing model for linguistic phenomena of interest (e.g. NER) is trained on intermediate representations of the encoder ([Tenney et al., 2019](#); [Kim et al., 2019](#)). Along similar lines, previous work on word embeddings looked for correlations between properties of the embeddings and downstream task performance ([Tsvetkov et al., 2016](#); [Rogers et al., 2018](#)). While interesting as analysis methods, these do not give users an understanding of how a fine-tuned (or end-to-end) model can handle linguistic phenomena *for the end-task*. For example, while [Tenney et al. \(2019\)](#) found that very accurate NER models can be trained using BERT (96.7%), we show BERT finetuned on QQP or SST-2 displays severe NER issues.

There are existing perturbation techniques meant to evaluate specific behavioral capabilities of NLP models such as logical consistency ([Ribeiro et al., 2019](#)) and robustness to noise ([Belinkov and Bisk, 2018](#)), name changes ([Prabhakaran et al., 2019](#)), or adversaries ([Ribeiro et al., 2018](#)). CHECKLIST provides a framework for such techniques to systematically evaluate these alongside a variety of other capabilities. However, CHECKLIST cannot be

directly used for non-behavioral issues such as data versioning problems ([Amershi et al., 2019](#)), labeling errors, annotator biases ([Geva et al., 2019](#)), worst-case security issues ([Wallace et al., 2019](#)), or lack of interpretability ([Ribeiro et al., 2016](#)).

6 Conclusion

While useful, accuracy on benchmarks is not sufficient for evaluating NLP models. Adopting principles from behavioral testing in software engineering, we propose CHECKLIST, a model-agnostic and task-agnostic testing methodology that tests individual *capabilities* of the model using three different test types. To illustrate its utility, we highlight significant problems at multiple levels in the conceptual NLP pipeline for models that have “solved” existing benchmarks on three different tasks. Further, CHECKLIST reveals critical bugs in commercial systems developed by large software companies, indicating that it complements current practices well. Tests created with CHECKLIST can be applied to any model, making it easy to incorporate in current benchmarks or evaluation pipelines.

Our user studies indicate that CHECKLIST is easy to learn and use, and helpful both for expert users who have tested their models at length as well as for practitioners with little experience in a task. The tests presented in this paper are part of CHECKLIST’s open source release, and can easily be incorporated into existing benchmarks. More importantly, the abstractions and tools in CHECKLIST can be used to collectively create more exhaustive test suites for a variety of tasks. Since many tests can be applied across tasks as is (e.g. typos) or with minor variations (e.g. changing names), we expect that collaborative test creation will result in evaluation of NLP models that is much more robust and detailed, beyond just accuracy on held-out data. CHECKLIST is open source, and available at <https://github.com/marcotcr/checklist>.

Acknowledgments

We would like to thank Sara Ribeiro, Scott Lundberg, Matt Gardner, Julian Michael, and Ece Kamar for helpful discussions and feedback. Sameer was funded in part by the NSF award #IIS-1756023, and in part by the DARPA MCS program under Contract No. N660011924033 with the United States Office of Naval Research.

References

- Saleema Amershi, Andrew Begel, Christian Bird, Rob DeLine, Harald Gall, Ece Kamar, Nachi Nagappan, Besmira Nushi, and Tom Zimmermann. 2019. [Software engineering for machine learning: A case study](#). In *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track*. IEEE Computer Society.
- Boris Beizer. 1995. *Black-box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., New York, NY, USA.
- Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*.
- Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.
- 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 *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1161–1166.
- Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of NAACL-HLT*, pages 1875–1885.
- Najoung Kim, Roma Patel, Adam Poliak, Patrick Xia, Alex Wang, Tom McCoy, Ian Tenney, Alexis Ross, Tal Linzen, Benjamin Van Durme, et al. 2019. Probing what different nlp tasks teach machines about function word comprehension. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019)*, pages 235–249.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Aakanksha Naik, Abhilasha Ravichander, Norman Sadeh, Carolyn Rose, and Graham Neubig. 2018. Stress Test Evaluation for Natural Language Inference. In *International Conference on Computational Linguistics (COLING)*.
- Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 667–676. ACM.
- 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.
- Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. [Know what you don’t know: Unanswerable questions for SQuAD](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.
- Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishal Shankar. 2019. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400.
- 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.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Association for Computational Linguistics (ACL)*.
- Anna Rogers, Shashwath Hosur Ananthkrishna, and Anna Rumshisky. 2018. [What’s in your embedding, and how it predicts task performance](#). In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2690–2703, Santa Fe, New Mexico, USA. 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.
- Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering*, 42(9):805–824.
- Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. [BERT rediscovered the classical NLP pipeline](#). In

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Yulia Tsvetkov, Manaal Faruqui, and Chris Dyer. 2016. [Correlation-based intrinsic evaluation of word vector representations](#). In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 111–115, Berlin, Germany. Association for Computational Linguistics.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. 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 2153–2162.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. [GLUE: A multi-task benchmark and analysis platform for natural language understanding](#). In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R’emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763.