Robustness Gym: Unifying the NLP Evaluation Landscape

Karan Goel[*†]
Stanford University

Nazneen Rajani[†]
Salesforce Research

Jesse Vig
Salesforce Research

Zachary Taschdjian
Salesforce Research

Mohit Bansal
UNC Chapel-Hill

Christopher Ré
Stanford University

Abstract

Despite impressive performance on standard benchmarks, natural language processing (NLP) models are often brittle when deployed in real-world systems. In this work, we identify challenges with evaluating NLP systems and propose a solution in the form of Robustness Gym (ℝ𝔾),[1] a simple and extensible evaluation toolkit that unifies 4 standard evaluation paradigms: subpopulations, transformations, evaluation sets, and adversarial attacks. By providing a common platform for evaluation, ℝ𝔾 enables practitioners to compare results from disparate evaluation paradigms with a single click, and to easily develop and share novel evaluation methods using a built-in set of abstractions. Robustness Gym is under active development and we welcome feedback & contributions from the community.

1 Introduction

Advances in natural language processing (NLP) have led to models that achieve high test accuracy on independent and identically distributed (i.i.d.) data. However, analyses suggest that models are not robust to data corruptions (Belinkov and Bisk, 2018), distribution shifts (Hendrycks et al., 2020; Miller et al., 2020), and harmful data manipulations (Jia and Liang, 2017), and rely on spurious patterns (McCoy et al., 2019b). In practice, these vulnerabilities hinder deployment of trustworthy systems, as seen in public-use systems that were later revealed to be systematically biased, such as chatbots (Stuart-Ulin, 2018) and recruiting tools (Hamilton, 2018).

While practitioners know of these problems, it remains common to evaluate solely on i.i.d. data. Ideally, the goal of evaluation is to perform a broad assessment of a model's capabilities on the types of examples that it is likely to see when deployed. This process is complex for practitioners, since existing tools cater to a specialized set of evaluations for a task, and provide no clear way to leverage or share findings from prior evaluations. Thus, current evaluation practices face two challenges:

1. **Idiomatic lock-in (Section 2.1).** We identify 4 distinct evaluation types or idioms supported by existing tools and research – subpopulations, transformations, adversarial attacks and evaluation sets. Existing tools use bespoke abstractions to serve a subset of these idioms (e.g., adversarial attacks on words), requiring users to glue together tools to perform a broad evaluation that mixes idioms.

2. **Workflow fragmentation (Section 2.2).** As practitioners evaluate, they need to save progress, report findings and collaborate to understand model behavior. Existing solutions to save progress are tool- and idiom-specific, lack versioning, and provide limited support for sharing. Existing reporting templates are free-form, and have not successfully incentivized users to report findings e.g. only 6% of Huggingface (Wolf et al., 2020b) models report evaluation information.

In response to these challenges, we introduce Robustness Gym (ℝ𝔾), a simple, extensible and unified toolkit for evaluating robustness and sharing findings (Figure 1). ℝ𝔾 users can:

1. **Create slices (Section 3.1)** of data in ℝ𝔾. Each slice is a collection of examples, built using one or more evaluation idioms. ℝ𝔾 scaffolds users in a two-stage workflow, separating the storage of side-information about examples (CachedOperation) from the nuts and bolts of programmatically building slices using this information (SliceBuilder). This workflow helps

[*] E-mail: kgoel@cs.stanford.edu
[†] Equal contribution.

[1] https://github.com/robustness-gym/robustness-gym

Proceedings of NAACL-HLT 2021: Demonstrations, pages 42–55
June 6–11, 2021. ©2021 Association for Computational Linguistics

	Type	Instantiation	Examples
Rule-based	Filters	HasPhrase HasLength Position	■ Subpopulation that contains negation. ■ Subpopulation that is in the `{X}` percentile for length. ■ Subpopulation that contains `{TOKEN}` in position `{N}`.
	Logic	IFTTT recipes Symmetry Consistency	■ If example ends in `{ING}` then transform with backtranslation. ■ Switch the first and last sentences of a source document to create a new eval set. ■ Adding "aaaabbbb" at the end of every example as a form of attack.
	Template	Checklist	■ Generate new eval set using examples of the form `"I {NEGATION} {POS_VERB}."`
Machine	Classifier	HasScore HasTopic	■ Subpopulation with perplexity `{>X}` based on a LM. ■ Subpopulation belonging to a certain topic.
	Tagger*	POS NER SRL Coref	■ Subpopulation that contains `{POS_NOUN}` in position `{N}`. ■ Subpopulation that contains entity names with non-English origin. ■ Subpopulation where there is no `{AGENT}`. ■ Subpopulation that contains the pronouns for a particular gender.
	Parser*	Constituency Dependency	■ Transform with all complete subtrees of `{POS_VP}` in the input. ■ Subpopulation that has at least 2 `{POS_NP}` dependent on `{POS_VP}`.
	Generative	Backtranslation Few-shot	■ Using a seq2seq model for transformation using backtranslation. ■ Using GPT3 like models for creating synthetic eval sets.
	Perturbation	Paraphrasing TextAttack	■ Synonym substitution using EDA. ■ Perturbing input using TextAttack recipes.
Human or Human-in-the-loop	Filtering	Figurative text	■ Using humans to identify subpopulation that contains sarcasm.
	Curation	Evaluation sets Data validation	■ Building datasets like ANLI, Contrast sets, HANS, etc. ■ Using human-in-the-loop for label verification.
	Adversarial	Invariant Directional	■ Perturbing text in a way that the expected output does not change. ■ Perturbing text in a way that the expected output changes.
	Transformation	Counterfactual	■ Transforming to counterfactuals for a desired target concept.

Table 1: Sample of slice builders and corresponding data slices along with example use cases that can either be used out-of-the-box or extended from Robustness Gym. ■ $\rightarrow$ subpopulations, ■ $\rightarrow$ transformations, ■ $\rightarrow$ adversarial attacks and ■ $\rightarrow$ evaluation sets. $*$ marked are CachedOperations and the rest are SliceBuilders.

users quickly implement new ideas, minimize boilerplate code and seamlessly integrate existing tools.

2. **Consolidate evaluations (Section 3.2)** and findings for community sharing. ℝ𝔾 users add slices into a TestBench that can be versioned and shared, allowing users to collaboratively build benchmarks and track progress. For standardized reporting, ℝ𝔾 provides *Robustness Reports* that can be auto-generated from testbenches and included in paper appendices.

We close with a discussion of how Robustness Gym can benefit practitioners[2] (Section 4), describing how users with varying expertise – novice, intermediate, expert – can evaluate a natural language inference (NLI) model in ℝ𝔾.

2 The Landscape of Evaluation Tools

We describe two challenges facing evaluation today, and situate them in the context of existing work.

[2]See 2 minute supplementary demo video.

2.1 Challenge 1: Idiomatic Lock-In

When practitioners decide what they want to evaluate, they can suffer from lock-in to a particular *idiom* or type of evaluation after they adopt a tool. Our analysis suggests that most tools and research today serve a subset of four evaluation idioms:

1. **Subpopulations.** Identify subpopulations of a dataset where the model may perform poorly.

   Example: short reviews (< 50 words) in the IMDB sentiment dataset (Maas et al., 2011).

2. **Transformations.** Perturb data to check that the model responds correctly to changes.

   Example: substitute words with their synonyms in the IMDB dataset.

3. **Attacks.** Perturb data adversarially to exploit weaknesses in a model.

   Example: add "aabbccaa" to the end of reviews, making the model predict positive sentiment.

4. **Evaluation Sets.** Use existing datasets or author examples to test generalization and perform targeted evaluation.

Evaluation Idiom	Tools Available	Research Literature (focusing on NLI)
Subpopulations	Snorkel (Ratner et al., 2017), Errudite (Wu et al., 2019)	Hard/easy sets (Gururangan et al., 2018) Compositional-sensitivity (Nie et al., 2019)
Transformations	NLPAug (Ma, 2019)	Counterfactuals (Kaushik et al., 2019), Stress test (Naik et al., 2018), Bias factors (Sanchez et al., 2018), Verb veridicality (Ross and Pavlick, 2019)
Attacks	TextAttack (Morris et al., 2020), OpenAttack (Zeng et al., 2020) Dynabench (Kiela, 2020)	Universal Adversarial Triggers (Wallace et al., 2019a), Adversarial perturbations (Glockner et al., 2018), ANLI (Nie et al., 2020)
Evaluation Sets	SuperGLUE diagnostic sets (Wang et al., 2019) Checklist (Ribeiro et al., 2020)	FraCaS (Cooper et al., 1994), RTE (Dagan et al., 2005), SICK (Marelli et al., 2014), SNLI (Bowman et al., 2015), MNLI (Williams et al., 2018), HANS (McCoy et al., 2019b), Quantified NLI (Geiger et al., 2018), MPE (Lai et al., 2017), EQUATE (Ravichander et al., 2019), DNC (Poliak et al., 2018), ImpPres (Jeretic et al., 2020), Systematicity (Yanaka et al., 2020) ConjNLI (Saha et al., 2020), SherLIiC (Schmitt and Schütze, 2019)

Table 2: Evaluation tools and literature, focusing on NLI as a case study. Some tools support multiple types of evaluations, e.g., TextAttack supports both augmentations and attacks. For additional related work, see Section 5.

Example: author new movie reviews in the style of a newspaper columnist.

We note that these idioms are not exhaustive. In Table 2, we use this categorization to summarize the tools and research available today, using the well-studied natural language inference (NLI) task as a case study. As an example, TextAttack (Morris et al., 2020) users can perform attacks, while CheckList (Ribeiro et al., 2020) users author examples using templates, but cannot perform attacks.

Tools vary in whether they provide scaffolding to let users build on new evaluation ideas easily. They often provide excellent abstractions for particular idioms, e.g., TextAttack (Morris et al., 2020) scaffolds users to easily write new adversarial attacks. However, no tool that we are aware of addresses this for evaluation that cuts across multiple idioms.

All of these limitations make it difficult for practitioners, who are forced to glue together a combination of tools. Each tool meets different developer needs, and has its own abstractions and organizing principles, which takes away time from users to inject their own creativity and expertise into the evaluation process.

We address these challenges with Robustness Gym (Section 3.1), which uses an open-interface design to support all 4 evaluation idioms, and provides a simple workflow to scaffold users.

2.2 Challenge 2: Workflow Fragmentation

As practitioners evaluate, they need to keep track of progress and communicate findings. Evaluation tools today let users save their progress, but provide no support for semantic versioning (Preston-Werner, 2013) and sharing findings. This is made more difficult when consolidating evaluations and results across multiple tools. General-purpose data storage solutions (McKerns et al., 2012) solve this problem, but require significant user effort to customize and manage.

Reporting findings can be difficult since there is no consensus on how to report when performing evaluation across multiple idioms. To study whether existing tools incentivize reporting, we scraped *model cards* for all available Huggingface models (Wolf et al., 2020a). Model cards (Mitchell et al., 2019) are free-form templates for standardized reporting that contain an entry for “Evaluation" or “Results", but leave the decision of what to report to the user. Huggingface provides tools for users to create model cards when submitting models to their model hub.

Our findings are summarized in Table 3. Only a small fraction (6.0%) of models carry model cards with any evaluation information. Qualitatively, we found low consistency in how users report findings, even for models trained on the same task. This suggests that it remains difficult for users to report evaluation information consistently and easily.

In Section 3.2, we describe the support that Robustness Gym provides for versioning evaluations in testbenches, and easily exporting and reporting findings with reports.

	# Model Cards	% of Models
Total	2133	64.6%
Non-empty	922	27.9%
Any evaluation info	197	6.0%
# Models	3301	100.0%

Table 3: Prevalence of evaluations in model cards on the HuggingFace Model Hub (huggingface.co/models).

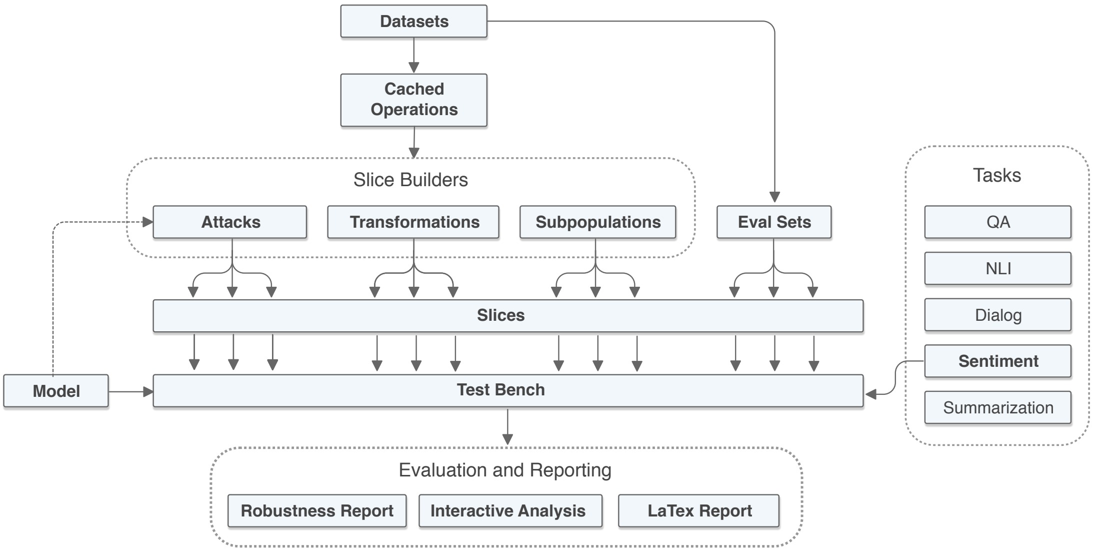

Figure 1: Robustness Gym system design and workflow.

3 Robustness Gym

We address the challenges highlighted in Section 2 with Robustness Gym (ℝ𝔾). We describe how users can build evaluations in Section 3.1, and version evaluations and report findings in Section 3.2. Figure 1 provides a visual depiction of the system design and workflow in ℝ𝔾, while Python examples for ℝ𝔾 are in Tables 4, 5 and 6 of the appendix.

3.1 Evaluation Workflow

As highlighted in Section 2.1, practitioners can get locked into a single tool that supports only a few evaluation idioms. By contrast, ℝ𝔾 enables broad evaluations across multiple idioms. At a high level, ℝ𝔾 breaks evaluation into a two-stage workflow:

1. **Caching information.** First, practitioners typically perform a set of common pre-processing operations (e.g., tokenization, lemmatization) and compute useful side information for each example (e.g., entity disambiguation, coreference resolution, semantic parsing) using external knowledge sources and models, which they cache for future analysis. An example is running the spaCy pipeline, and caching the Doc object that is generated for downstream analysis.

   A large part of practitioner effort goes into generating this side information – which can be expensive to compute – and into standardizing it to a format that is convenient for analysis.

   RG Support. CachedOperation is an abstraction in ℝ𝔾 to derive useful information or generate side information for each example in a dataset by (i) letting users run common operations easily and caching the outputs of these operations e.g., running spaCy (Honnibal et al., 2020); (ii) storing this information alongside the associated example so that it can be accessed conveniently; (iii) providing a simple abstraction for users to write their own operations.

2. **Building slices.** Second, practitioners use the examples' inputs and any available cached information to build *slices*, which are collections of examples used for evaluation based on any of the 4 evaluation idioms. These slices are derived from a loaded dataset by applying one of the evaluation idioms, e.g. filtering a dataset based on some criteria to construct a subpopulation.

   RG Support. SliceBuilder is an abstraction to retrieve information for an example and create slices of data from them by (i) providing retrieval methods to access inputs and cached information conveniently when writing custom code to build slices; (ii) providing specialized abstractions for specific evaluation idioms: transformations, attacks and subpopulations.

   Robustness Gym includes wrappers for libraries such as TextAttack and nlpaug that provide specialized support for constructing adversarial attacks and data transformations respectively. This allows users the ability to utilize external libraries in a unified toolkit and workflow.

This breakdown naturally separates the process of gathering useful information from the nuts and

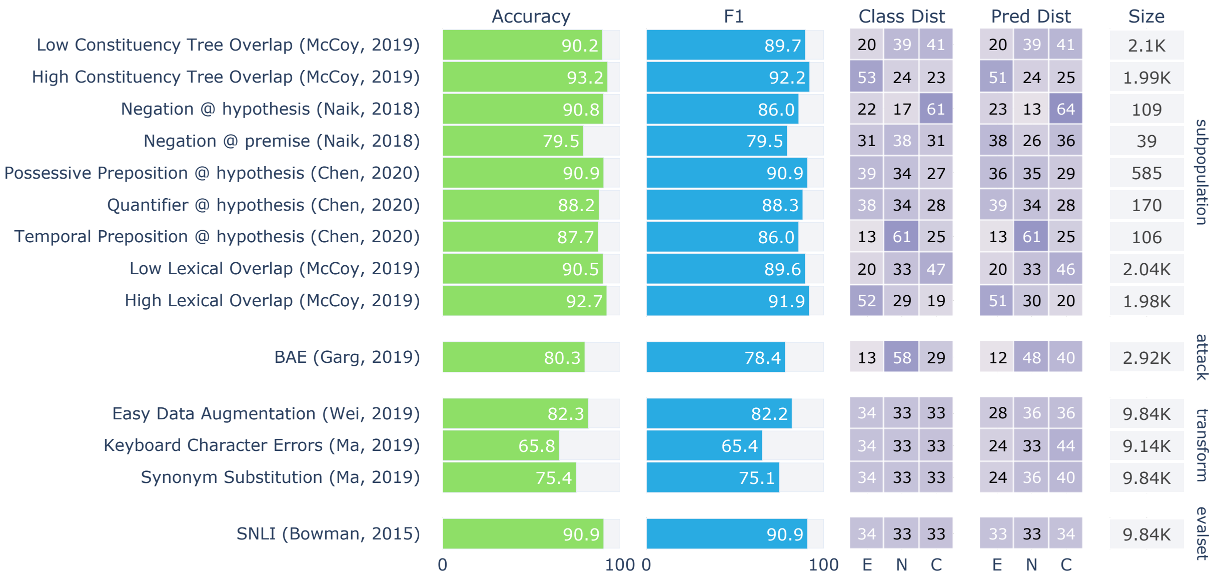

Figure 2: Robustness Report for Natural Language Inference using `bert-base` on SNLI.

bolts of using that information to build slices. Table 1 contains examples of CachedOperations and SliceBuilders that can be supported by ℝ𝔾.

ℝ𝔾 relies on a common data interface provided by the datasets library from HuggingFace (Wolf et al., 2020a), which is backed by Apache Arrow (Foundation, 2019). This ensures that all operations in ℝ𝔾 interoperate with HuggingFace models.

3.2 Testbenches and Reports

As highlighted in Section 2.2, users may find themselves consolidating evaluation results across several tools and evaluation idioms. ℝ𝔾 addresses this fragmentation by providing users a TestBench abstraction for storing and versioning evaluations, and a Report abstraction for sharing findings.

- **Versioning evaluations.** Users can assemble and version a collection of slices into a TestBench, which represents a suite of evaluations. A TestBench contains the slices created by the user, and users can interact with a TestBench to evaluate models and store metrics. Each TestBench has an associated semantic version that can be "bumped" as changes are made, e.g. if a user adds a new set of slices, they can change the version to indicate that the TestBench has been modified.

  ℝ𝔾 tracks the provenance or history of slices, making it easy to identify the (i) slice's original data source; (ii) sequence of SliceBuilders by which a slice was created. This makes it easy for another user to reproduce evaluations when given a TestBench, even without the original code. They can simply inspect the slices in the TestBench to look at provenance information, and use it to reproduce their evaluation process.

- **Sharing findings.** Users can create a *Robustness Report* for any model on a TestBench (Figure 2), or standalone reports for evaluations that are not performed in ℝ𝔾, using the Report abstraction. To incentivize standardized reporting, ℝ𝔾 supports *Standard Reports* for several tasks. The Standard Report is comprehensive, static and is backed by a TestBench that contains slices from all evaluation idioms. It can be generated in a PDF or LaTeX format to be added to the appendix of a paper[3]. Reports reduce user burden in communicating findings, and make it easier to standardize reporting in the community.

  ℝ𝔾 supports an interactive Streamlit tool[4] for generating standard reports, which will be expanded in the future to allow users to pick slices based on their evaluation needs.

4 User Personas in Robustness Gym

Next, we discuss how users with varying expertise can use ℝ𝔾. We describe how 3 user personas—beginner, intermediate, and advanced—can use ℝ𝔾

[3] See Figure 3 in the appendix.
[4] Screenshot in Figure 4 of the appendix.

to analyze the performance of an natural language inference (NLI) model. In NLI, the goal is to determine whether a premise sentence entails, is neutral to, or contradicts a hypothesis sentence.

4.1 Scenario I: Beginner User

Description. Users new to NLP and robustness, lack knowledge to choose or write specific slices.

Example Goal. Exploratory robustness testing.

RG support:

- *Visual Interface.* The user creates a report with a few clicks in the Streamlit interface[5]. They select "Standard Report", "SNLI" (dataset)[6], "Ternary Natural Language Inference" (task), "BERT-Base" (model), and click "Generate Report".
- *Standard Reports.* The Standard Report, shown in Figure 2 provides a detailed snapshot of various robustness tests for NLI. The tests may include Subpopulations (e.g., HasNegation, LexicalOverlap), Transformations (e.g., SynonymAug, KeyboardAug) (Ma, 2019), Attacks (TextAttack) (Morris et al., 2020; Garg and Ramakrishnan, 2020), and Evaluation Sets (Bowman et al., 2015). The user gleans several initial insights from this report. For example, their model is vulnerable to typing mistakes due to low accuracy on the KeyboardAug slice; the predicted class distribution column reveals that this noise causes the model to predict `contradiction` more frequently than `entailment` or `neutral`. The user is able to easily share the generated PDF of this report.

4.2 Scenario II: Intermediate User

Description. Users familiar with NLP and robustness, willing to write minimal code.

Example Goal. Explore gender bias when gendered pronouns are present in the input.

RG support:

- *Built-in SliceBuilders.* Apply the existing HasPhrase SliceBuilder to create subpopulations with female pronouns in the hypothesis:

```
subpopulations = HasPhrase(['her', 'she'])
slices = subpopulations(snli, ['hypothesis'])
```

- *Testbenches.* Put slices into a TestBench and make it available on GitHub for collaboration.
- *Reports.* Generate Robustness Reports for any model from the TestBench.

4.3 Scenario III: Advanced User

Description. NLP experts, need to write custom code for their task and research.

Example Goal. Evaluate whether NLI models rely on surface-level spurious similarities between premise and hypothesis.

RG support:

- *CachedOperations.* Run the spaCy pipeline for tokenization.
- *Custom SliceBuilders.* Utilize the ScoreSubpopulation class to construct subpopulations with arbitrary scoring functions. Write a custom scoring function `len_diff` that returns the absolute difference in length between the tokenized hypothesis and premise. Then, find examples that score in the top 10% as follows:

```
s = ScoreSubpopulation(
   intervals=[('90%','100%')], score_fn=len_diff)
```

- *Transformations.* Transform data using classes such as EasyDataAugmentation (Wei and Zou, 2019). Compose this transformation with the custom ScoreSubpopulation described earlier to create a larger slice.
- *Testbench.* Publish a new TestBench on GitHub for others to reuse and refine the evaluations.
- *Report.* Generate a report for immediate analysis and a LaTeX appendix to share results in a research paper (see Figure 3 in appendix).

5 Related Tools and Work

We highlight additional related work for evaluation and reporting, including work on interpretability.

Evaluation and error-analysis. Tools for evaluation and error analysis support users in understanding where their models fail. In contrast to RG, existing tools support only a subset of evaluations and analyses. Errudite (Wu et al., 2019), Snorkel (Ratner et al., 2017) support subpopulations, TextAttack (Morris et al., 2020) adversarial attacks, nlpaug (Ma, 2019) transformations, and CrossCheck (Arendt et al., 2020), Manifold (Zhang et al., 2018) focus on visualization and analysis for model comparison.

Interpretability. Tools for interpretability enable a better understanding of model behavior. These

[5] See supplementary demo video for example usage.

[6] The Stanford Natural Languge Inference dataset (Bowman et al., 2015).

tools serve complementary objectives to Robustness Gym, e.g., explaining why a model makes a certain prediction, rather than performing broad evaluations. Tools include the recent Language Interpretability Tool (LIT) (Tenney et al., 2020), IBM's AI Explainability 360 (Arya et al., 2019), AllenNLP Interpret (Wallace et al., 2019b), InterpretML (Nori et al., 2019), Manifold (Zhang et al., 2018), Pytorch Captum (Narine Kokhlikyan and Reblitz-Richardson), DiCE (Mothilal et al., 2020), What-if (Wexler et al., 2019), FairVis (Cabrera et al., 2019), and FairSight (Ahn and Lin, 2019). Many of these tools focus on interactive visualization, which limits their scope to interpreting small numbers of examples and makes their use susceptible to subjectivity and selection bias. By contrast, Robustness Gym can scale to large datasets, while testbenches ensure reproducibility of analyses.

6 Conclusion

We introduced Robustness Gym, an evaluation toolkit that supports a broad set of evaluation idioms, and can be used for collaboratively building and sharing evaluations and results. Robustness Gym is under active development and we welcome feedback and contributions from the community.

Acknowledgements

This work was part of a collaboration between Stanford, UNC, and Salesforce Research and was supported by Salesforce AI Research grants to MB and CR. We are thankful to Samson Tan, Jason Wu, Stephan Zheng, Caiming Xiong, Han Guo, Laurel Orr, Jared Dunnmon, Chris Potts, Marco Tulio Ribeiro, Shreya Rajpal for helpful discussions and feedback. CR also gratefully acknowledges the support of NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), and 1937301 (RTML); ONR under No. N000141712266 (Unifying Weak Supervision); the Moore Foundation, NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, the Okawa Foundation, American Family Insurance, Google Cloud, Swiss Re, Total, the HAI-AWS Cloud Credits for Research program, and members of the Stanford DAWN project: Facebook, Google, and VMWare. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of NIH, ONR, or the U.S. Government.

References

Yongsu Ahn and Yu-Ru Lin. 2019. Fairsight: Visual analytics for fairness in decision making. *IEEE transactions on visualization and computer graphics*, 26(1):1086–1095.

Dustin Arendt, Zhuanyi Huang, Prasha Shrestha, E. Ayton, Maria Glenski, and Svitlana Volkova. 2020. Crosscheck: Rapid, reproducible, and interpretable model evaluation. *ArXiv*, abs/2004.07993.

Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. 2019. One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques.

Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. *ArXiv*, abs/1711.02173.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. 2019. Fairvis: Visual analytics for discovering intersectional bias in machine learning. In *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 46–56. IEEE.

Vincent Chen, Sen Wu, Alexander J Ratner, Jen Weng, and Christopher Ré. 2019. Slice-based learning: A programming model for residual learning in critical data slices. In *Advances in neural information processing systems*, pages 9392–9402.

Robin Cooper, Dick Crouch, Jan Van Eijck, Chris Fox, Johan Van Genabith, Jan Jaspars, Hans Kamp, David Milward, Manfred Pinkal, Massimo Poesio, and et al. Pulman, Stephen. 1994. Using the framework. Technical report, Deliverable D6.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Apache Software Foundation. 2019. Arrow: A cross-language development platform for in-memory data.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. *ArXiv*, abs/2004.01970.

Atticus Geiger, Ignacio Cases, Lauri Karttunen, and Christopher Potts. 2018. Stress-testing neural models of natural language inference with multiply-quantified sentences. *arXiv preprint arXiv:1810.13033*.

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.

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

Isobel Asher Hamilton. 2018. Amazon built an AI tool to hire people but had to shut it down because it was discriminating against women.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. 2020. The many faces of robustness: A critical analysis of out-of-distribution generalization. *arXiv preprint arXiv:2006.16241*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Paloma Jeretic, Alex Warstadt, Suvrat Bhooshan, and Adina Williams. 2020. Are natural language inference models imppressive? learning implicature and presupposition. *arXiv preprint arXiv:2004.03066*.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *EMNLP*.

Divyansh Kaushik, Eduard Hovy, and Zachary C Lipton. 2019. Learning the difference that makes a difference with counterfactually-augmented data. *arXiv preprint arXiv:1909.12434*.

Douwe Kiela. 2020. Rethinking AI Benchmarking.

Alice Lai, Yonatan Bisk, and Julia Hockenmaier. 2017. Natural language inference from multiple premises. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 100–109, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Edward Ma. 2019. NLP Augmentation.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. SemEval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 1–8, Dublin, Ireland. Association for Computational Linguistics.

R. T. McCoy, Ellie Pavlick, and Tal Linzen. 2019a. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *ArXiv*, abs/1902.01007.

R Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019b. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*.

M. McKerns, Leif Strand, T. Sullivan, Alta Fang, and M. A. G. Aivazis. 2012. Building a framework for predictive science. *ArXiv*, abs/1202.1056.

J. Miller, Karl Krauth, B. Recht, and L. Schmidt. 2020. The effect of natural distribution shift on question answering models. *ArXiv*, abs/2004.14444.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229.

John X Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks in natural language processing. *arXiv preprint arXiv:2005.05909*.

Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 607–617.

Aakanksha Naik, Abhilasha Ravichander, Norman Sadeh, Carolyn Rose, and Graham Neubig. 2018. Stress test evaluation for natural language inference. *arXiv preprint arXiv:1806.00692*.

Miguel Martin Edward Wang Jonathan Reynolds Alexander Melnikov Natalia Lunova Narine Kokhlikyan, Vivek Miglani and Orion Reblitz-Richardson. Pytorch captum.

Yixin Nie, Yicheng Wang, and Mohit Bansal. 2019. Analyzing compositionality-sensitivity of nli models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6867–6874.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial nli: A new benchmark for natural language understanding. In *ACL*.

Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. 2019. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*.

Adam Poliak, Aparajita Haldar, Rachel Rudinger, J. Edward Hu, Ellie Pavlick, Aaron Steven White, and Benjamin Van Durme. 2018. Collecting diverse natural language inference problems for sentence representation evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 67–81, Brussels, Belgium. Association for Computational Linguistics.

Tom Preston-Werner. 2013. Semantic versioning 2.0. 0. *línea]. Available: http://semver. org*.

Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access.

Abhilasha Ravichander, Aakanksha Naik, Carolyn Rose, and Eduard Hovy. 2019. EQUATE: A benchmark evaluation framework for quantitative reasoning in natural language inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 349–361, Hong Kong, China. 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 *Association for Computational Linguistics (ACL)*.

Alexis Ross and Ellie Pavlick. 2019. How well do nli models capture verb veridicality? 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 2230–2240.

Swarnadeep Saha, Yixin Nie, and Mohit Bansal. 2020. Conjnli: Natural language inference over conjunctive sentences. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8240–8252.

Ivan Sanchez, Jeff Mitchell, and Sebastian Riedel. 2018. Behavior analysis of NLI models: Uncovering the influence of three factors on robustness. 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 1975–1985, New Orleans, Louisiana. Association for Computational Linguistics.

Martin Schmitt and Hinrich Schütze. 2019. SherLIiC: A typed event-focused lexical inference benchmark for evaluating natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 902–914, Florence, Italy. Association for Computational Linguistics.

Chloe Rose Stuart-Ulin. 2018. Microsoft's politically correct chatbot is even worse than its racist one.

Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, et al. 2020. The language interpretability tool: Extensible, interactive visualizations and analysis for nlp models. *arXiv preprint arXiv:2008.05122*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019a. Universal adversarial triggers for nlp. *arXiv preprint arXiv:1908.07125*.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019b. Allennlp interpret: A framework for explaining predictions of nlp models. *arXiv preprint arXiv:1909.09251*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3266–3280.

Jason W Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*.

James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2019. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. 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 1112–1122. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020a. Huggingface's transformers: State-of-the-art natural language processing.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020b. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tongshuang Wu, Marco Tulio Ribeiro, J. Heer, and Daniel S. Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *ACL*.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, and Kentaro Inui. 2020. Do neural models learn systematicity of monotonicity inference in natural language? *arXiv preprint arXiv:2004.14839*.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2020. Openattack: An open-source textual adversarial attack toolkit. *arXiv preprint arXiv:2009.09191*.

Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. 2018. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373.

A Appendix

Code. We provide example code snippets for Robustness Gym in Tables 4 (CachedOperation), 5 (SliceBuilder), and 6 (TestBench, Report), below.

LaTeX Report. Figure 3 is an example of a report generated in a LaTeX format. The code for the figure was auto-generated and the figure was simply included in the appendix.

Streamlit Application. Figure 4 is a screenshot of our streamlit application for generating standard reports.

Goal			Code Snippet
Caching	Create	Create Spacy cached operation	`spacy = Spacy()`
		Create Stanza cached operation	`stanza = Stanza()`
		Create a custom cached operation	`cachedop = CachedOperation( apply_fn=my_custom_fn, identifier=Identifier('MyCustomOp'), )`
		Run a cached operation	`dataset = cachedop(dataset, columns)`
	Retrieve	Retrieve all Spacy info cached	`Spacy.retrieve(dataset, columns)`
		Retrieve Spacy tokens	`Spacy.retrieve(batch, columns, 'tokens')`
		Retrieve Stanza entities	`Stanza.retrieve( batch, columns, Stanza.entities )`
		Retrieve any cached operation info after processing	`CachedOperation.retrieve( batch, columns, my_proc_fn, 'MyCustomOp' )`

Table 4: Code for the CachedOperation abstraction in Robustness Gym.

Goal			Code Snippet

Slice Building	Subpopulations	Create a subpopulation that generates three slices based on raw lengths in $[0, 10]$, $[10, 20]$ and $[20, \infty)$	`length_sp = Length(`
			`    [(0, 10), (10, 20), (20, np.inf)]`
			`)`

		Create a subpopulation that generates two slices based on bottom 10% and top 10% length percentiles	`length_sp = Length(`
			`    [('0%', '10%'), ('90%', '100%')]`
			`)`

		Create a custom subpopulation by binning the outputs of a scoring function	`custom_sp = ScoreSubpopulation(`
			`    [('0%', '10%'), ('90%', '100%')],`
			`    my_scoring_fn`
			`)`

	Transformations	Create EasyDataAugmentation	`eda = EasyDataAugmentation()`

		Create any NlpAug transformation	`nlpaug_trans = NlpAugTransformation(`
			`    pipeline=nlpaug_pipeline`
			`)`

		Create a custom transformation	`custom_trans = Transformation(`
			`    Identifier('MyTransformation'),`
			`    my_transformation_fn`
			`)`

	Attacks	Create TextAttack recipe	`attack = TextAttack.from_recipe(recipe, model)`
	Evaluation Sets	Create a slice from a dataset	`sl = Slice(dataset)`

	Slice Builders	Run any SliceBuilder	`dataset, slices, membership = slicebuilder(`
			`    batch_or_dataset=dataset,`
			`    columns=columns,`
			`)`

Table 5: Code for the SliceBuilder abstraction in Robustness Gym.

Goal			Code Snippet

Reporting	Testbench	Create a testbench	`testbench = TestBench(`
			`    identifier=Identifier('MyTestBench'),`
			`    version='0.1.0'`
			`)`

		Add slices to testbench	`testbench.add_slices(slices)`
		Fuzzy search testbench for slices	`top_k_matched_slices = testbench.search('len')`
		Bump testbench minor version	`testbench.bump_minor()`
		Save and load a testbench	`testbench.save(path)` `testbench.load(path)`
	Report	Evaluate model on slices and generate report	`testbench.create_report(model)`

		Create a custom report	`report = Report(`
			`    dataframe_with_metrics,`
			`    report_columns,`
			`)`

		Generate figure from report	`figure = report.figure()`
		Generate LaTeXreport	`latex = report.latex()`

Table 6: Code for the TestBench and Report abstractions in Robustness Gym.

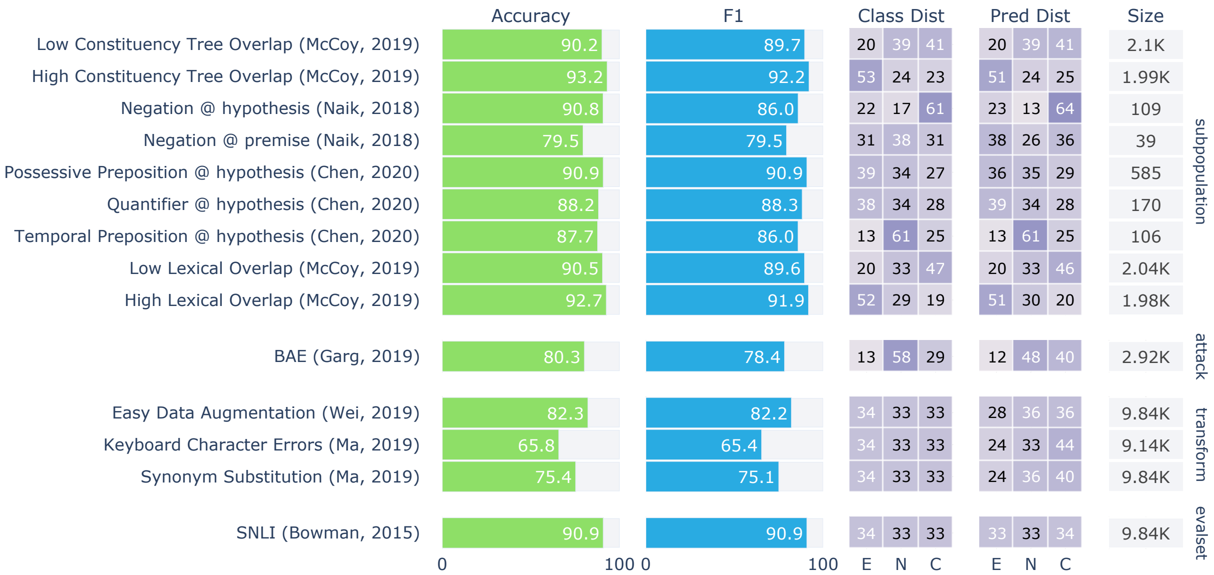

Figure 3: Robustness report for textattack/bert-base-uncased-snli model on SNLI dataset. The report lays out scores for each evaluation, broken out by category. Citations: (Chen et al., 2019; Naik et al., 2018; McCoy et al., 2019a; Wei and Zou, 2019; Ma, 2019; Bowman et al., 2015).

Note: the LaTeX figure and caption above is auto-generated using "report.latex()".

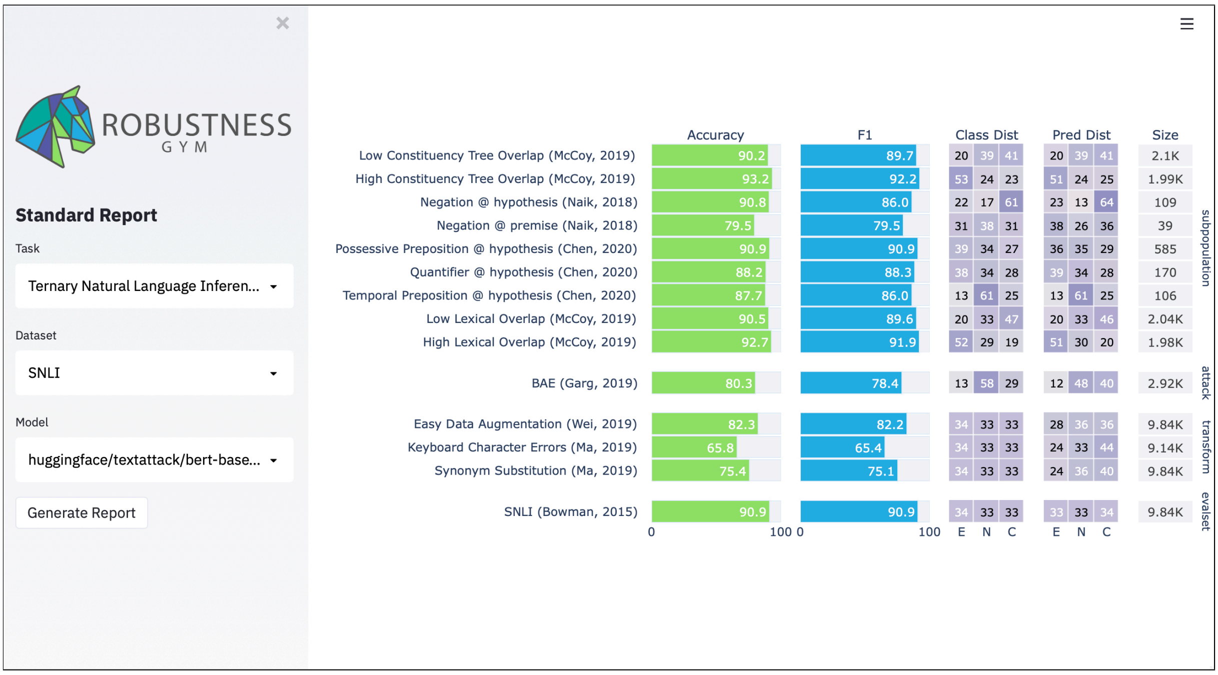

Figure 4: Screenshot of our interactive Streamlit application for creating standard reports. Users can choose a task, dataset and model on the left side, and a standard report spanning all 4 evaluation idioms – subpopulations, transformations, attacks and evaluation sets – is auto-generated on the right side.