Capabilities for Better ML Engineering

Chenyang Yang[1], Rachel Brower-Sinning[2], Grace A. Lewis[2], Christian Kästner[1] and Tongshuang Wu[1]

[1]*School of Computer Science, Carnegie Mellon University*

[2]*Carnegie Mellon Software Engineering Institute*

Abstract

In spite of machine learning's rapid growth, its engineering support is scattered in many forms, and tends to favor certain engineering stages, stakeholders, and evaluation preferences. We envision a capability-based framework, which uses fine-grained specifications for ML model behaviors to unite existing efforts towards better ML engineering. We use concrete scenarios (model design, debugging, and maintenance) to articulate capabilities' broad applications across various different dimensions, and their impact on building safer, more generalizable and more trustworthy models that reflect human needs. Through preliminary experiments, we show the potential of capabilities for reflecting model generalizability, which can provide guidance for the ML engineering process. We discuss challenges and opportunities for the integration of capabilities into ML engineering.

Keywords
machine learning engineering, capability, specification, testing, evaluation

1. Introduction

Despite the rapid evolution of machine learning models, most effort has been on *prototyping* models — developing models under idealized settings (e.g., with static datasets, following the i.i.d. assumption, assuming equal importance of all mistakes). These models tend to suffer in the wild where the ideal assumptions do not hold, leading to safety issues, fairness issues, and project failures [1]. For example, a pedestrian detection model trained on images taken on sunny days would not correctly respond to natural weather changes [2] and may have never seen a wheelchair user in training or test data. Oversimplification has real consequences. If we had only tested the aforementioned pedestrian detector on similar, sunny test examples, and used our overly optimistic evaluation to support deployment decisions, then an automated vehicle with the detector would be likely to cause accidents.

To actually integrate models into production, substantial additional engineering effort is required by interdisciplinary teams [3]: Not only do we need to make careful decisions at the model level (e.g., develop evaluation metrics that reflect human expectations on models [4]), but we also need to connect the model with the broader system design (e.g., the model functionalities should be well-specified in a *requirements engineering process* [5], similar to how we design user interfaces).

The importance of these efforts, commonly referred to as *ML engineering* [6], has been well-recognized, but the actual implementation tends to be scattered. For example, academic research on ML engineering tends to focus on the narrow space of model testing and debugging for data scientists [e.g., 7, 8], whereas industrial efforts are mostly limited to supporting pipeline automation and model deployment ("MLOps") [9]. More importantly, because these efforts are isolated, it is unclear how insights from one stage can be transferred to benefit the entire ML engineering process (e.g., how error analysis results help update model design decisions). In other words, there is still a lack of synergy among existing efforts for better ML engineering practices.

In this work, we envision a unified framework for ML engineering. In particular, we center our framework around *capabilities* [4]. A capability is a form of fine-grained specification for ML model behavior. It helps define concrete model behaviors in various scenarios which are finer-grained and more holistic than standard evaluation metrics. In our pedestrian detector example, different capabilities can be used to express safety requirements from different aspects, e.g., recognizing pedestrians in wheelchairs, being robust to extreme weather, or being fair to people from different age groups [2].

Similar to other ML engineering efforts, the term capability emerged specifically from (and is mostly used in) model testing and debugging [4, 8]. However, its natural link with *expected model behaviors* makes it ideal for *ML model specification* which, akin to software specification, (1) builds the root for the entire ML engineering cycle, going from model design all the way to deployment and maintenance, and (2) serves as the *boundary object* [10] for different stakeholders to negotiate their (sometimes conflicting) expectations of models. Moreover, capabilities have the potential to reflect multiple essential factors in ML engineering, e.g., distribution shift [11], robustness [12], fairness [13] (see Tab. 1). However, capabilities

Table 1
Example capabilities for pedestrian detection models. Capabilities commonly express what a human would expect from ML models (common knowledge, robustness, human-style reasoning) and can reflect different model qualities (generalizability, robustness, fairness). We also illustrate possible instantiation strategies to produce concrete examples from capabilities.

Capability	Instantiation	Origin/Theory
Recognize pedestrians in wheelchairs	Curate images w/ pedestrians in wheelchairs	Knowledge of important outliers
Robust to extreme weather	Transform sunny images to rainy	Robustness to anticipated distribution shift
Detect pedestrians of all ages	Slice test data by pedestrian age	Reasoning about concept variations

have yet to fulfill their potential due to several challenges, e.g., it is not clear how to (1) best identify capabilities, (2) instantiate abstract capabilities, and (3) operationalize capabilities to maximize their utility.

We take the first step towards presenting the vision of a capability-based framework that both unites existing efforts and sheds light on future opportunities. Specifically, we illustrate the broad applicability of the framework from both the technical perspective and the practical perspective, by (1) summarizing how existing ML engineering concepts can be expressed with capabilities, and (2) describing four usage scenarios with unique characteristics (model debugging, collaboration, external quality assurance, and model maintenance). We also conduct an exploratory study to demonstrate the feasibility of our vision. We conclude the paper by discussing challenges and opportunities for capabilities' integration into ML engineering that emerge from our preliminary results.

2. Capabilities

Capability definition: ML "specification." A capability can roughly be defined as a fine-grained specification of behaviors expected of an ML model. The key idea is to go beyond just considering the overall accuracy of a model but analyzing to what degree the model exhibits specific kinds of expected behaviors. The term capability was popularized by work on testing specific behaviors of ML models [4], but similar concepts can be found in other work on model testing (e.g., stress tests [14]) and in various work exploring nuances of model misbehavior and shortcut learning (e.g., underspecifications [15]). Previous work [e.g., 4, 8] has shown that capabilities can expose many systematic problems in state-of-the-art models, are useful for interactive testing and debugging, and can guide data augmentation to train better models.

Capabilities share similarities with traditional *software specifications* (and *functional requirements*) in that both prescribe how software should behave in specific scenarios. These prescriptions are general concepts or descriptions but can be concretized into a list of input-output examples (i.e., test cases) for assessing models in the engi-

neering process. We refer to the process of deriving test data from capabilities as *instantiation*. Capabilities can be instantiated in many different ways, including slicing existing data [7], transformation of existing data [16], generating data from templates [4], and targeted curation of new data (possibly with crowdsourcing) [17] – see examples in Tab. 1. Different instantiation strategies have different costs and benefits, and it is often necessary to make trade-offs between them.

However, capabilities also differ from traditional specifications in fundamental ways: Traditional software is built using a *deductive reasoning* process. Their specifications are usually hard rules the software must satisfy – a single input-output pair that violates the specification will be considered a bug. In contrast, machine learning uses *inductive reasoning*, where models are derived from observations and are expected to make occasional mistakes [18]. As such, instead of declaring a model as buggy for a single mistake related to a capability, we measure *to what degree* the model has certain capabilities with a *failure rate*. In this sense, capability can be viewed as a soft *lower bound specification*, and we use failure rates to look for issues where a model systematically underperforms with regard to a capability.

Capabilities as a unifying framework. There are many existing efforts to support ML engineering, but they are often scattered and unconnected. Evaluating models on specific qualities like robustness, fairness, and generalizability is extensively discussed [e.g., 19, 13, 20], but they often focus exclusively on a narrow set of capabilities (e.g., robust to word replacement [21], data shift [11], and spurious correlations [22]). Different strategies for model evaluation and data augmentation, from slicing [7], counterfactuals [17, 23, 24], templates [4], to perturbations [16] are widely explored, but there are very little efforts on combining them, evaluating their relative costs and effectiveness, and often such efforts are limited to individual qualities (e.g., robustness [12]). Recent work has shown interest in model debugging [8, 25] and error analysis [7], but they often use different terminologies despite the similar underlying ideas.

Table 2
Example usage scenarios for capabilities. These scenarios cover different ML engineering stages and stakeholders, showing capabilities are beneficial across dimensions.

Scenario	Stages	Stakeholders
Model Debugging	Development	Data Scientists
Collaboration	Requirements, Evaluation	Software Engineers, Data Scientists
External QA	Evaluation	External Evaluators, Regulators
Model Maintenance	Deployment	Data Scientists, End Users

We argue that a *capability* is a generic abstraction that can unify existing efforts. For example, different model evaluation strategies can be seen as ways to instantiate capabilities; different model qualities can be viewed as (a series of) capabilities that might matter in specific scenarios; a model's reliance on spurious correlations can be interpreted as a lack of specific capabilities (e.g., ignoring backgrounds for object detection [26]). Furthermore, as we will argue, capabilities can go beyond existing literature to benefit engineering stages (e.g., requirements engineering) and stakeholders (e.g., external evaluators or software engineers) that are currently under-explored.

3. Capabilities for Better ML Engineering

ML engineering effort happens at different development **stages**, with different **stakeholders** in the loop, and targets different model **qualities**. We argue that capabilities can help unify ML engineering efforts and lead to more systematic practice because they can play important roles in all these diverse dimensions.

Below, we describe four concrete ML engineering scenarios (summarized in Tab. 2), which cover different dimensions and highlight challenges and opportunities.

3.1. Illustrative Scenarios

Scenario 1: Model Debugging. Alice is a data scientist responsible for a chatbot used in her company. She is now debugging the conversational model that performs poorly on some inputs. She tries to understand what is going wrong with these model mistakes. For each mistake, she speculates the potential issue behind it (e.g., input sentence contains numerical reasoning that the current model does not handle well) and updates the model accordingly. However, she finds the entire process ad-hoc and does not always produce a better model.

Capabilities can systematize this process and help Alice generalize from individual mistakes to systematic problems. Instead of chasing mistakes, Alice now *identifies* common capabilities from model mistakes. Then she *assesses* the importance of different capabilities, *instantiates* the prioritized ones, and uses the instantiated tests for both training and evaluation. Alice now evaluates the new model not only on some general test data, but also on the test suites of different capabilities. She finds that the new model handles numerical reasoning better but is slightly worse on a different test suite that requires complex co-reference resolution. She decides that this is acceptable and releases the model.

Scenario 2: Collaboration. Bob is a software engineer working in a government department, dealing with classified information. The department has a contract with an external data science team on a vision model for satellite images, which is expected to be robust to various attacks and stable across various environments. Due to strict data security policies, the external data science team relies on public datasets instead of actual production data. Bob struggles to communicate requirements and report useful feedback when the delivered model does not work in production.

Capabilities can serve as a *communication* interface between different stakeholders. Bob would be able to clearly describe the failures in ways the data science team can understand, if he abstracts concrete private data, and identifies sharable *capabilities* from them. Or even better, he can instantiate capabilities with public data points, such that the data science team can develop the next version of the model with a clear goal of improvement in mind in terms of capability failure rates.

Scenario 3: External Quality Assurance. Carolyn works for a quality assurance team that previously focused on testing traditional software components. Carolyn is now responsible for independently evaluating models delivered by external contractors — this time a model for fraud detection. Trained in traditional software testing, Carolyn finds it challenging to move forward without concrete specifications at hand, and is unsure what to do beyond standard accuracy evaluations.

Capabilities provide a more holistic view of how models perform in different scenarios. Carolyn reuses known capabilities for fraud detection, which her team developed for assessments on previous models, and evaluates the model on instantiated test suites from these capabilities, diving into specific capabilities of the model rather than providing just a single broad accuracy measure. She also looks at production data and past mistakes, and uses them to *identify* new capabilities. Her final report *communicates* how the model performs on different capability

test suites and highlights the model's major weaknesses.

Scenario 4: Model Maintenance.   Dan is a data scientist for a social media platform. They are responsible for a model that detects toxicity from user posts. The model performs well on previously curated data, but its performance degrades over time because of evolving trends in user posts. Dan tries to update the model periodically to cope with data shift. However, they find that the model is still frequently suboptimal to unknown future shifts even when trained with more recent data.

Capabilities can be used to track how data evolves through time and characterize data shift. Dan now maintains a list of high-quality capability test suites as regression tests. They regularly review new data to *identify* whether the model needs additional capabilities, or whether the reliance on existing capabilities changes over time. This way, Dan gets to track the capability shift trajectory, anticipate (to some extent) what future shift might look like, and can *instantiate* suitable capabilities tests beforehand. With capabilities, Dan now builds and selects models that are more robust to data shift.

Discussion.   We described four different scenarios of using capabilities for better ML engineering, illustrating their broad applicability. As a recap,

- Capabilities can be used at different **stages** of ML engineering. On the one hand, they provide *specifications* for ML models, which is fundamental to (collaborative) model design, development, and testing. On the other hand, they also provide valuable abstractions for concrete data points, serve as a form of data specification, and allow for characterizing (possibly changing) deployment environments. Notably, this potential for data documentation/specification further enlarges capabilities' impact on various stages that concern data, e.g., data collection, dataset evaluation, etc.
- Different **stakeholders** can utilize capabilities. Though data scientists, external evaluators, etc. in our scenarios have different priorities in mind, they are able to converge on the capability framing — whether to use capabilities to exploit their hypotheses on model mistakes, to communicate the characteristics of a non-shareable deployment environment, or to utilize prior training practices. Notably, as in the communication case, such convergence enables knowledge sharing or even negotiation between stakeholders, as everyone can speak the same "language."
- Capabilities can relate to different **qualities** of ML models, ranging from accuracy (e.g., in *debugging*), robustness (e.g., in *collaboration*), fairness, to generalizability (e.g., in *maintenance*). This enables multifaceted evaluation without more consistent metric designs, which is valuable especially when multiple

Table 3

Capabilities and their instantitation keywords for sentiment analysis, selected based on existing work [27]. We slice the validation data on keywords to instantiate these capabilities, and the % column represents the ratio of validation data that is included in the slice.

Capability	%	Keywords
negation	51.6	not, n't
negation *(v2)*	18.7	no, never, neither, nobody, none, nor, nothing
shifter	4.5	refuse, reject, deny, doubt, abandon, miss, question, abort, stop
modality	3.6	would have, could have, should have
comparative	16.6	better, worse, than
mixed	36.4	but, however, though, although, despite, even if, rather than, except that
reducer	14.1	kind of, all that, less, a little, somewhat, still
amplifier	48.8	really, very, super, so, incredibly, extremely, at all, whatsoever, much

model qualities have to be balanced.

Despite the promising future, these scenarios share common challenges, from identifying, assessing, communicating, to instantiating capabilities. Yet different scenarios focus on different aspects and might have different requirements for the same challenge. For example, all scenarios require identifying capabilities, but the ways they are identified or expressed vary; a *shared language* would be required for collaboration, but if different stakeholders describe the same capabilities in different ways, or have different instantiation ideas, then additional inconsistency arises and has to be mitigated. We will discuss these practical barriers in the next section.

3.2. Exploratory Experiment

To explore the practicality of our envisioned capability framework, we conducted an experiment to explore whether capabilities are reflective of model generalizability. We focus on generalizability first because it is a primary design goal for any ML model, and a model quality essential for various use scenarios (e.g., the aforementioned model maintenance and collaboration).

Experiment setup.   We define "reflective" as the statistical correlation between model performance on certain capability tests, and their performance on out-of-distribution data points. [1]

Specifically, in the experiment, we repeatedly fine-tuned BERT with different random seeds on the AMAZON-WILDS dataset [20], and obtained 100 sentiment analysis models with similar source domain accuracy (Amazon

---

[1] Experiment details can be found in an online appendix (https://github.com/malusamayo/Capabilities-Experiment-Details) and are not essential for the main vision outlined in this paper.
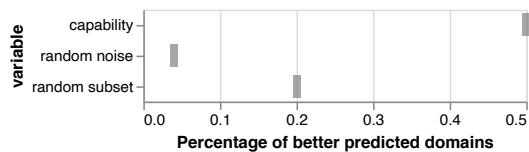
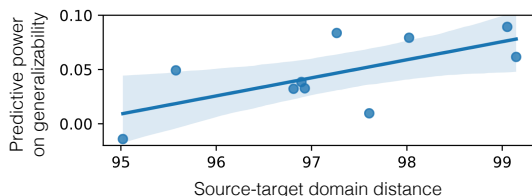**Figure 1:** Capabilities better help predict model generalization compared to other baselines.

Figure 2: Predictive power improvement correlates with distribution distance. The further the distribution is, the better capabilities could help predict generalization. We hypothesize that this is because if a target distribution is too close to the source distribution, there is little room left for improvement.

product reviews on HOME-AND-KITCHEN) but different target domain accuracy on 10 domains (e.g., MOVIE-AND-TV reviews). We also selected eight capabilities for sentiment analysis from an existing study [27], and instantiated them into test suites through data slicing, as in Tab. 3. For each target domain, we fit a linear model to find correlations between the models' target domain accuracy (dependent variable), and the models' source domain accuracy, as well as their capability testing results (independent variables). We looked at adjusted R^2 to see whether the model has a better fit when these test results are part of the independent variable, compared to otherwise (i.e., whether these extra variables help predict out-of-distribution accuracy).

Results. *Model performance on capability tests is a strong signal for model's generalizbability.* We can confirm results from prior capability-testing experiments: Even fairly generic capabilities are somewhat helpful in predicting how well models generalize to out-of-distribution data. In Fig. 1, on 50% of the target domains (5/10), having capability tests adds a significant signal on models' generalizability to the target domain (i.e., significantly higher adjusted R^2). In contrast, baselines with model performances on randomly sliced subsets or random noise do not provide similar improvement.

Capability tests especially helps predict how well models generalize to further domains. We also mapped out the approximated distances between each target domain and the source domain, using a proxy $\mathcal{A}$-distance [28]. As in

Fig. 2, we observe a positive slope between the distance and the R^2 power. This shows that capabilities are particularly helpful for distributions that deviate more from the training distribution, such as in Bob's scenario where distribution details could not be shared.

Discussion. Besides the positive signals, our experiment also highlighted several challenges we faced when using capabilities during ML engineering. In particular, we observe that vanilla capability identification and instantiation have limited utility, for two reasons:

- *Different capabilities add different amount of information.* Some capabilities (e.g., *negation*) produce too many test cases, which leads to an uninformative distribution close to the source dataset, while others (e.g., *modality*) result in a rather distinct distribution, which is more informative for predicting generalization.
- *Different capabilities add different kinds of information.* Some capabilities are complementary but others are highly correlated and add little additional information over other capabilities. For example, in our experiment, we found that using only *shifter* improves predicting generalizability in 20% cases, but adding *modality* further improves in 40% cases. At the same time, capabilities could also add conflicting information, where models perform or generalize worse if they better support a capability, which is similar to common tradeoffs between model accuracy and other qualities (e.g., robustness).

In essence, the design space of capabilities and their corresponding instantiations is massive. While prior work has reported positive impacts of capabilities, as well as success in scaffolding the identification and instantiation process [4], few studies have comprehensively evaluated the information gain of different capabilities, the interactions between capabilities, and the effectiveness of different identification / instantiation strategies.

In our experiment, we resolved to the most basic and typical methods for identification and instantiation, which has inherent limitations: We identified capabilities by reusing domain knowledge from existing work, which is not tailored for generalization to specific target distribution; we instantiated capabilities through coarse-grained slicing on keywords, which does not always produce useful test suites (e.g., *negation*). While we also considered other identification and instantiation strategies, we eventually discarded them as they require much more manual effort — a reflection on the reality that most people would probably prefer simpler (if rather flawed) methods.

As a result, we argue that proper guidance needs to be designed, such that different stakeholders can quickly climb the rather steep learning curve for making capabilities useful. We discuss future directions next.

4. Challenges and Opportunities

To more systematically use capabilities, further research is needed. We argue that ML engineering can generally benefit from software engineering disciplines, with principles from requirements engineering and software testing in particular. In the following, we identify promising research directions based on gaps in the literature and our own observations in our experiment.

Identifying capabilities. It is challenging to identify capabilities for concrete scenarios. Capabilities often differ across different modes (vision vs. language), different tasks (sentiment analysis vs. natural language inference), and different domains (product reviews vs. book reviews). While we may develop a catalog of common capabilities for general-purpose tasks, such as sentiment analysis [27], we will likely need to identify specific capabilities for each domain-specific problem. Existing strategies include using domain knowledge [16], performing error analysis [14, 7, 25], and mining knowledge from existing corpora [29]. Most strategies require extensive efforts of domain experts or crowdsource workers, making them hard to scale. They are also often conducted in an unsystematic way and do not draw on classic requirement elicitation and participatory design approaches. Future work could explore:

RQ1 How could we support more effective discovery and reuse of domain knowledge? When and how can we automate discovery?

RQ2 What kinds of mechanisms could support more efficient human-AI interaction in error analysis?

RQ3 How could we design a better process to help both experts and non-experts identify capabilities?

Assessing capabilities. Capabilities often exhibit a hierarchical structure. For example, *understanding negation* is a very general capability, whereas *understanding double negation* or *handling modifiers* as "hardly" and "never" are more specific (sub-)capabilities. How fine-grained a capability should be will likely depend on the specific scenarios. More coarse capabilities are more reusable, whereas finer-grained ones capture concrete concepts that might be especially useful for the domain (but may not transfer — e.g., concrete adjectives like "cold" is positive when describing refrigerators but not so much for thermos). Their predictiveness also differs across scenarios, as we observed in our experiments. When identifying capabilities, we need to determine the proper granularity, and evaluate their importance within the context:

RQ4 What is a good granularity for a capability?

RQ5 How do we evaluate/rank capabilities by context?

Communicating capabilities. Identified capabilities need to be efficiently communicated between different stakeholders, who might have different requirements and potential conflicts, or may describe the same capabilities in drastically different ways depending on their expertise (e.g., an expert may say "invariant to environmental conditions" when a lay user says "performs the same in sunny, raining, stormy weathers.") Common communication vocabularies and conflict resolution mechanisms, possibly informed by existing requirements engineering literature, would greatly facilitate the process.

RQ6 How can we develop a shared language or interface to facilitate capability communication?

RQ7 How can capabilities support conflict resolution between different stakeholders?

Instantiating capabilities. Abstract capabilities need to be instantiated as concrete test cases, to be further used as regression tests, examples for communication, or augmentation data for training. Existing work has explored different strategies for instantiating capabilities (c.f. Sec. 2), but it remains unclear how different strategies perform in different scenarios and whether they could be combined in a meaningful way. These strategies are similar to software testing (e.g., unit tests and metamorphic testing [30]) and can be informed by existing software engineering literature (e.g., test case generation, fuzzing, prioritization, and requirements validation).

RQ8 How should we select instantiation strategies in different scenarios? How to measure and trade off costs and benefits?

RQ9 How do different instantiation strategies complement each other?

5. Conclusion

A capability is a generic abstraction that unifies existing efforts on model testing, debugging, and evaluation. It can also benefit the entire ML engineering lifecycle from data collection to model deployment, addressing the needs of different stakeholders and model qualities. Our exploratory experiments showed that capabilities could provide strong signals for model generalizability, as well as highlighted challenges in integrating them into the ML engineering process. We hope future research will better support identifying, assessing, communicating, and instantiating capabilities.

Acknowledgments

Institute, a federally funded research and development center (DM22-1187).

References

[1] K. Panetta, Gartner identifies the top strategic technology trends for 2021. (2020).

[2] D. Gerónimo, A. M. López, A. D. Sappa, T. Graf, Survey of pedestrian detection for advanced driver assistance systems, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (2010) 1239–1258.

[3] N. Nahar, S. Zhou, G. Lewis, C. Kästner, Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process, in: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 2022, pp. 413–425.

[4] M. T. Ribeiro, T. Wu, C. Guestrin, S. Singh, Beyond accuracy: Behavioral testing of NLP models with CheckList, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 4902–4912.

[5] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, 1st ed., Wiley Publishing, 2009.

[6] A. Burkov, Machine learning engineering, volume 1, True Positive Incorporated, 2020.

[7] T. Wu, M. T. Ribeiro, J. Heer, D. Weld, Errudite: Scalable, reproducible, and testable error analysis, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 747–763.

[8] M. T. Ribeiro, S. Lundberg, Adaptive testing and debugging of NLP models, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 3253–3267.

[9] S. Mäkinen, H. Skogström, E. Laaksonen, T. Mikkonen, Who needs mlops: What data scientists seek to accomplish and how can mlops help?, 2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN) (2021) 109–112.

[10] S. L. Star, The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989, p. 37–54.

[11] S. Rabanser, S. Günnemann, Z. C. Lipton, Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift, Curran Associates Inc., Red Hook, NY, USA, 2019.

[12] K. Goel, N. F. Rajani, J. Vig, Z. Taschdjian, M. Bansal, C. Ré, Robustness gym: Unifying the NLP evaluation landscape, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations, Association for Computational Linguistics, Online, 2021, pp. 42–55.

[13] D. S. Shah, H. A. Schwartz, D. Hovy, Predictive biases in natural language processing models: A conceptual framework and overview, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 5248–5264.

[14] A. Naik, A. Ravichander, N. Sadeh, C. Rose, G. Neubig, Stress test evaluation for natural language inference, in: Proceedings of the 27th International Conference on Computational Linguistics, Association for Computational Linguistics, Santa Fe, New Mexico, USA, 2018, pp. 2340–2353.

[15] A. D'Amour, et al., Underspecification presents challenges for credibility in modern machine learning, 2020.

[16] K. D. Dhole, et al., Nl-augmenter: A framework for task-sensitive natural language augmentation, 2021.

[17] D. Kaushik, E. Hovy, Z. Lipton, Learning the difference that makes a difference with counterfactually-augmented data, in: International Conference on Learning Representations, 2020.

[18] C. Kaestner, Machine learning is requirements engineering — on the role of bugs, verification, and validation in machine learning, Blog, 2020.

[19] J. Ebrahimi, D. Lowd, D. Dou, On adversarial examples for character-level neural machine translation, in: Proceedings of the 27th International Conference on Computational Linguistics, Association for Computational Linguistics, Santa Fe, New Mexico, USA, 2018, pp. 653–663.

[20] P. W. Koh, et al., Wilds: A benchmark of in-the-wild distribution shifts, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 5637–5664.

[21] Z. Sun, J. M. Zhang, Y. Xiong, M. Harman, M. Papadakis, L. Zhang, Improving machine translation systems via isotopic replacement, in: Proceedings of the 44th International Conference on Software Engineering, ICSE '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 1181–1192.

[22] T. McCoy, E. Pavlick, T. Linzen, Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference, in: Proceedings of the 57th Annual Meeting of the Association for Computa-

tional Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 3428–3448.

[23] M. Gardner, et al., Evaluating models' local decision boundaries via contrast sets, in: Findings of the Association for Computational Linguistics: EMNLP 2020, Association for Computational Linguistics, Online, 2020, pp. 1307–1323.

[24] T. Wu, M. T. Ribeiro, J. Heer, D. Weld, Polyjuice: Generating counterfactuals for explaining, evaluating, and improving models, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Online, 2021, pp. 6707–6723.

[25] A. A. Cabrera, A. J. Druck, J. I. Hong, A. Perer, Discovering and validating ai errors with crowd-sourced failure reports, Proc. ACM Hum.-Comput. Interact. 5 (2021).

[26] S. Beery, G. Van Horn, P. Perona, Recognition in terra incognita, in: Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI, Springer-Verlag, Berlin, Heidelberg, 2018, p. 472–489.

[27] J. Barnes, L. Øvrelid, E. Velldal, Sentiment analysis is not solved! assessing and probing sentiment classification, in: Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, Association for Computational Linguistics, Florence, Italy, 2019, pp. 12–23.

[28] J. Blitzer, M. Dredze, F. Pereira, Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification, in: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Association for Computational Linguistics, Prague, Czech Republic, 2007, pp. 440–447.

[29] H. Barzamini, M. Rahimi, M. Shahzad, H. Alhoori, Improving generalizability of ml-enabled software through domain specification, in: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI, CAIN '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 181–192.

[30] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, Z. Q. Zhou, Metamorphic testing: A review of challenges and opportunities, ACM Comput. Surv. 51 (2018).