Discovering and Validating AI Errors With Crowdsourced Failure Reports

ÁNGEL ALEXANDER CABRERA, Carnegie Mellon University, USA
ABRAHAM J. DRUCK, Carnegie Mellon University, USA
JASON I. HONG, Carnegie Mellon University, USA
ADAM PERER, Carnegie Mellon University, USA

AI systems can fail to learn important behaviors, leading to real-world issues like safety concerns and biases. Discovering these systematic failures often requires significant developer attention, from hypothesizing potential edge cases to collecting evidence and validating patterns. To scale and streamline this process, we introduce crowdsourced *failure reports*, end-user descriptions of how or why a model failed, and show how developers can use them to detect AI errors. We also design and implement *Deblinder*, a visual analytics system for synthesizing failure reports that developers can use to discover and validate systematic failures. In semi-structured interviews and think-aloud studies with 10 AI practitioners, we explore the affordances of the *Deblinder* system and the applicability of failure reports in real-world settings. Lastly, we show how collecting additional data from the groups identified by developers can improve model performance.

CCS Concepts: • **Human-centered computing** → *Visual analytics*; *Social tagging*; • **Computing methodologies** → **Machine learning**.

Additional Key Words and Phrases: machine learning; crowdsourcing; debugging; blind spots; visual analytics

ACM Reference Format:
Ángel Alexander Cabrera, Abraham J. Druck, Jason I. Hong, and Adam Perer. 2021. Discovering and Validating AI Errors With Crowdsourced Failure Reports. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 425 (October 2021), 22 pages. https://doi.org/10.1145/3479569

1 INTRODUCTION

AI systems deployed in the real world can have significant consequences when they fail, including safety issues like self-driving cars harming pedestrians [62], and fairness concerns like inadvertently racist image labels [58]. Developers need to be able to efficiently detect, validate, and fix systematic failures to improve the safety, equity, and overall performance of their systems [48].

Unfortunately, discovering what systematic errors AI systems make presents developers with various challenges [2, 25, 31]. First, developers often have to sift through thousands of failure cases, many of which are random, one-off errors, to identify systematic failures. This manual task can be prohibitively labor-intensive for individuals or small teams. Second, once a developer discovers a pattern of failure, they have to validate their hypothesis by finding more evidence of the same

Authors' addresses: Ángel Alexander Cabrera, cabrera@cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, USA, 15213; Abraham J. Druck, adruck@andrew.cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, USA, 15213; Jason I. Hong, jasonh@cs.cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, USA, 15213; Adam Perer, adamperer@cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, USA, 15213.

2573-0142/2021/10-ART425
https://doi.org/10.1145/3479569

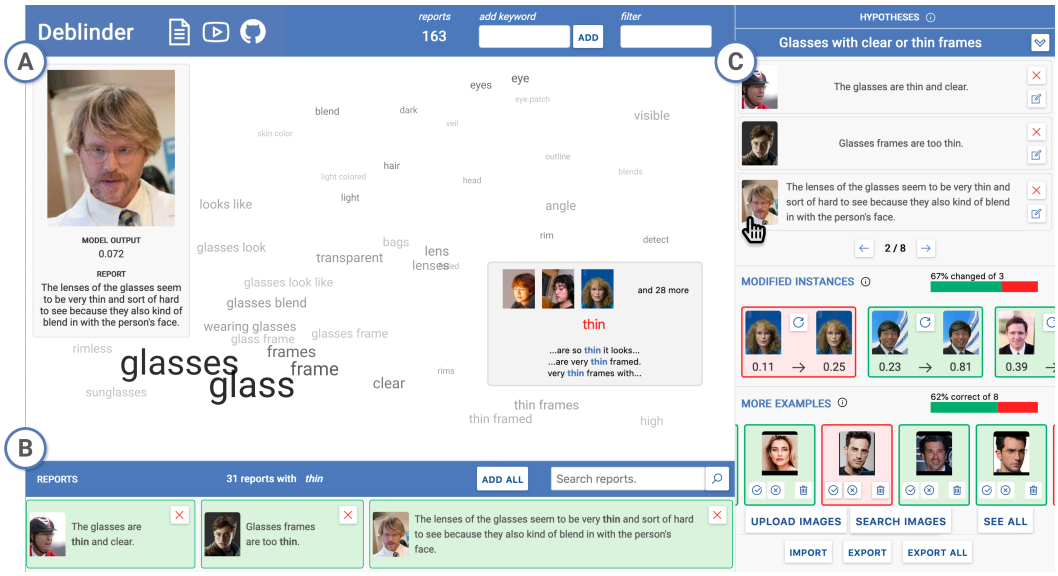

Fig. 1. The *Deblinder* system looking at failure reports for a classification model that detects if a person is wearing eyeglasses. The descriptions of why each instance was misclassified are textual reports generated by crowdworkers. In the Failure Report Embedding **(A)**, a developer can explore high-level concepts extracted from the failure reports. They can then search for or look through specific reports using the Failure Report Drawer **(B)**. Finally, the developer can create hypotheses for blind spots in the Hypothesis Panel **(C)** and test them by modifying instances or collecting additional data.

behavior. In common domains like images, there are few tools for finding similar instances, making it hard to find more evidence. Lastly, data scientists have to repeat this process to track and mitigate the various systematic failures a model may have.

To tackle these challenges, we define and formalize crowdsourced *failure reports*, text descriptions from end-users detailing how or why they think an AI system failed. Failure reports can, for example, describe a face recognition model that didn't detect a person who was outside, or a smartwatch that miscounted steps when a user was running on a treadmill. When collected at scale in a *crowd auditing* process, failure reports can be used to discover AI errors developers were unaware of and provide evidence to validate their hypotheses. Failure reports are inspired by the well-established concept of bug reports in software engineering [8], but are different in a few significant ways. Primarily, since AI systems are often stochastic, black-box models, failure reports have to be aggregated and further validated to uncover AI failures. In this work, we discuss the similarities and differences between failure reports and bug reports and the design of methods for collecting useful reports. We also tested different collection methods and techniques and implemented an example report collection system using Amazon's Mechanical Turk platform.

Failure reports can describe AI failures, but collecting hundreds or thousands of free-text sentences leads us back to one of data scientists' original challenges: they still have to sift through countless failure reports to find and validate patterns. To address this issue, we designed and implemented a visual analytics system, *Deblinder*, that lets users explore hundreds or thousands of failure reports to discover and test systematic failures. The system's main interface is a visualization that aggregates failure reports to let developers find patterns of error. Developers can then create hypotheses for failures they find in the report visualization. *Deblinder* also provides

two complementary features for validating failure hypotheses and assessing if they generalize - similar instance search and instance manipulation. While *Deblinder* is focused on image models, it is designed to be adapted to other domains like text and video data.

We explored the applications of failure reports and the *Deblinder* system through semi-structured interviews and think-aloud studies with 10 AI practitioners. We found that the process of creating and testing hypotheses for systematic failures mirrors developers' debugging process and that they found consistent failures with supporting evidence when using *Deblinder*. The study also scoped the best uses of failure reports, including their limitation to domains where end-users can understand the input data and see the model output directly. Lastly, we experimentally showed that a model retrained with failures discovered by study participants can improve model performance.

Failure reports add a useful strategy to developers' AI debugging toolbox. They complement existing algorithmic, visual, and crowd debugging systems by detecting and describing complex failures in deployment, like those developers may not have considered due to their own blind spots and biases [6, 25, 49, 56]. Visualizing crowdsourced failure reports continues the emerging theme of distributed or crowd sensemaking [20, 21, 23, 32, 33], which has been used to improve clustering [3, 13], summarize bug reports [27], and learn model features [17].

In summary, our contributions are the following:

- ***Failure reports*, end-user descriptions of how or why an AI system failed.** We formalize a crowd auditing process for discovering AI failures based on *failure reports*, text descriptions of model errors from end-users. We explore the parallels between failure reports and software bug reports, design methods for effectively collecting them, and implement an example collection method using Amazon Mechanical Turk.
- ***Deblinder*, a visual analytics system for making sense of failure reports.** We designed and implemented a visual analytics system for synthesizing failure reports developers can use to discover and validate AI failures. The system uses an interactive word-embedding visualization to aggregate and spatially organize the text reports. Developers can then create, track, and test hypotheses using *Deblinder*, which provides two validation methods.
- **Evaluation of failure reports and the *Deblinder* system.** In semi-structured interviews and think-aloud studies with 10 AI practitioners, we explored the real-world applications and limitations of failure reports. We found that developers discovered consistent, evidence-backed failures using *Deblinder*, and showed experimentally that collecting data from the discovered failures improved model performance.

2 BACKGROUND AND RELATED WORK

Bug reports in software engineering. Bug reports are an essential stage of the software engineering process. A bug report generally consists of a description of the issue, steps to reproduce the problem, and supporting information like a stack trace [8, 66], which software engineers can use to find and fix the failing code. While at a high level failure reports are similar to bug reports, debugging AI systems present various new challenges. Primarily, while a single bug report can be used to identify and fix a bug, various examples of the same issue are required to discover a systematic failure in an AI system. Additionally, there is no ground-truth evidence like a stack trace [55] from faulty code in black-box AI systems - the only supporting evidence comes from failure reports and instances themselves. Therefore, developers need numerous failure reports for the same issue and additional evidence to detect and validate AI failures.

While failure reports differ significantly from bug reports, we build on existing software engineering research on improving bug reporting. Bug summaries can be helpful for quickly triaging bugs and finding similar issues [28, 50]. For example, one method for improving reports showed

how using crowd-elicited attributes could improve bug summaries [27]. Visualizations, like the topic modeling approach by Yeasmin et al. [65], have also been used to improve bug reporting by summarizing bug repositories. Lastly, finding duplicate bugs is also an active area of interest for reducing bug reports and finding more evidence for an issue [9, 59, 60]. While we do not summarize or remove duplicate failure reports, we extract keywords and combine similar reports to provide a high-level view of the issues end-users describe.

Error analysis for AI systems. We use the term *systematic failures* to describe a group of instances sharing semantic features for which an AI produces the wrong output significantly more often than for the overall dataset. Since there is no standard term in the literature for *systematic failures*, we use it interchangeably with terms from existing work like *blind spots* [49] or *unknown unknowns* [5, 36].

There are various *algorithmic techniques* for discovering and characterizing systematic failures, ranging from fully automated to human-driven, crowdsourced techniques. Fully algorithmic strategies aim to automatically slice data to discover areas of error, for example Lakkaraju et al.'s exploration-exploitation technique for clustering and discovering model blind spots [36] and Slice Finder, which splits data according to tabular features and uses the model's loss to rank their severity [18]. These techniques require data that can be easily clustered or sliced, both of which are rarely available for data like images and videos. By using human-generated failure reports, developers can find more nuanced and complex failures that are not defined by pre-existing features.

In addition to algorithmic methods, many *visualization tools* exist for helping data scientists develop and debug AI systems. These tools provide support across the entire ML development pipeline, from model tuning to error characterization. For example, Squares [53] and MLCube [29] are visualizations for tracking models' performance across different dimensions, including class confusion and various performance metrics. Visualization tools can also help developers debug models for specific types of error, for example, the What-If Tool [61], FairSight [1], and FairVis [11] are visual analytics systems for auditing the fairness of AI systems. These techniques, like the algorithmic methods, are limited to errors described using input features or model outputs. By using text reports, our method can describe any human-defined failure.

There are also visual systems specifically for exploring and characterizing model *errors*. Errudite is one such system specific for natural language processing (NLP) models [64]. It uses a regex-based querying language for searching and replacing parts of text to discover and correct error hypotheses. This technique works well for text data but requires users to already have hypotheses and does not generalize to domains like images. AnchorViz is a polar-coordinates visualization that lets users create semantic anchors to visualize data instances across different concepts [15]. This method requires users to manually label sentences with 'anchors' or concepts, which does not scale easily. Failure reports can surface initial hypotheses and supporting evidence for systematic failures.

Lastly, while not specifically designed for error analysis, data programming can be a powerful tool for discovering slices, or subgroups, of a dataset [24, 51]. Data programming is a method of combining noisy labeling functions to train a classifier. Developers can create labeling functions to quickly slice their data and discover systematic failures, similar to how MLCube [29] and Slice Finder [18] help developers do subgroup analysis. Data programming can be a helpful validation tool for discovering more evidence for errors detected using failure reports.

Crowd auditing. Using crowdsourced human input has shown promise for discovering and characterizing AI models' systematic failures. The first study to show that humans could effectively find AI failures was Beat the Machine, a fully human-driven technique that asked users to find examples for which an AI system failed, specifically for hate speech detection [4]. They found that humans could quickly find websites that the AI misclassified with very high probability. However,

the authors did not take on the subsequent problem of aggregating the individually reported errors to describe and validate a model's systematic failures.

Crowds can also be used to establish the boundaries of AI behavior and prevent models from making harmful decisions. Mandel et al. [39] explored how to use the crowd to define acceptable AI behavior, created a rule-based interface to generate AI system constraints, and showed its efficacy in a real-world education domain. *Deblinder* and failure reports can help identify the edge cases that would inform this type of crowd deliberation.

Recent work has explored how crowd input can be combined with algorithmic techniques to characterize AI failures. Nushi et al. [42] developed Pandora, a system that uses human and machine described clusters of data to derive a decision tree visualization of model errors. Pandora is an effective method for finding features that correlate with or predict failure, visualized with a usable decision tree. Like the clustering method by Lakkaraju et al. [36], Pandora is dependent on the clustering algorithm to find meaningful groups of failures. By using free text input, failure reports can describe nuanced failures that may not be found with clustering.

Liu et al. [37] introduced another method for detecting systematic failures, Patterned Beat the Machine (P-BTM), that extends BTM by using crowd workers to find more examples of unknown unknowns. They ask crowd workers to provide initial labels for unknown unknowns, which are then used to train an *expansion classifier* to find more examples in an unlabeled dataset. Like clustering methods, P-BTM requires a classification model for any semantic concept, which they note is often challenging. It also requires labeling initial unknown unknowns using a mechanism like BTM. Crowd auditing with failure reports uses the human labels to discover the initial failure instances *and*, with a system like *Deblinder*, surface similar instances that may be difficult for the expansion classifier to find.

Pandora and P-BTM are effective methods for detecting systematic failures that complement failure reports and *Deblinder* well. Failure reports are better suited for finding semantic failures not present in a training or test set. Since failure reports are text sentences, they can describe failures that involve actions or arbitrary semantic features, including those that are challenging to detect with a clustering or classification algorithm. When collected for deployed systems, failure reports can also detect real-world failures that are not present in a test set and labeled as an unknown unknown. Pandora and P-BTM, on the other hand, can be more efficient and require less manual analysis since they partially automate the discovery and validation of systematic failures. Developers can collect failure reports and find initial systematic failures with *Deblinder* that can be further validated using these hybrid, algorithmic methods.

3 CROWD AUDITING WITH FAILURE REPORTS

3.1 Motivating Scenario

To motivate the use of failure reports for detecting systematic failures and describe the process, we walk through an example scenario.

Kristen, an ML developer, has created an eyeglass detection system for use in airport customs. For security reasons, people should not wear glasses when taking their picture at passport control. Given a picture of someone's face, the system provides a binary output of whether or not the person is wearing glasses. While the system has a high test set accuracy, Kristen wants to know if her model performs well in practice.

At each photo kiosk, Kristen adds a text prompt that lets people write a report if the model fails to detect if they're wearing glasses or not. After a month, she has collected thousands of failure reports, and uses *Deblinder* to explore the reports and discover any systematic issues. With this information, she collects additional data representative of those blind spots and retrains her model.

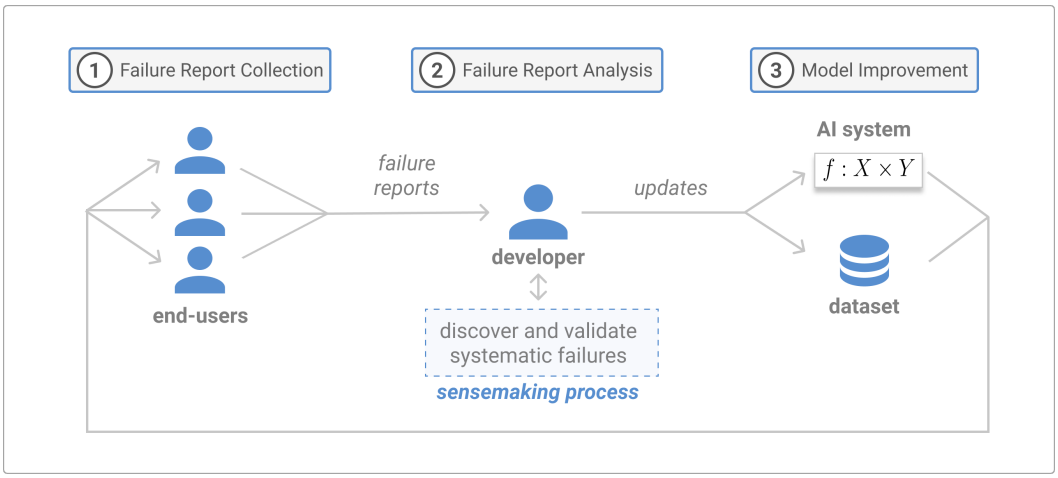

Fig. 2. The iterative process of using failure reports to discover and mitigate AI models' systematic failures. **(1)** In the **Failure Report Collection** stage, end users observe and report model failures back to the developer. **(2)** Next, in the **Failure Report Analysis** stage, developers analyze the failure reports to discover, describe, and validate hypotheses of potential failures. In this work, *Deblinder* is used as the primary interface for this sensemaking process. **(3)** In the final **Model Improvement** stage, developers can use the discovered failures to collect additional data, tweak their models, and analyze the real-world impact of their systems.

The retrained model performs better on the test set, and she deploys the updated system. She continues collecting reports to see if the blind spots persist or new issues arise over time.

We collect failure reports for the domain in this example, eyeglass detection, and use the results in our user study in Section 6. Specifically, we trained a convolutional neural network (CNN) using PyTorch to classify headshots of people, with an overall accuracy of over 99%. We trained the model using images from the CelebA dataset, which has over 160,000 headshots of celebrities with labels for eyeglasses [38]. Additionally, we collected reports and applied *Deblinder* to the domain of image captioning (see Section 9.1) to show how the process and system generalize to other domains.

3.2 Process Overview

The crowd auditing process with failure reports consists of three major stages, as can be seen in Figure 2. It is an iterative and ongoing process, as end-users report newly discovered failures and developers update their model. The three main stages are the following: (1) Failure Report Collection, (2) Failure Report Analysis, and (3) Model Improvement.

In the first stage, **Failure Report Collection**, end-users of an AI system describe why or how a model failed for a given instance. This stage can either be done during development to proactively find potential blind spots or be conducted in deployment to discover real-world problems. The second stage is **Failure Report Analysis**, where developers have to make sense of the numerous text failure reports. In the last stage, **Model Improvement**, developers can use their validated insights to mitigate the real-world effects of model errors and improve their systems.

Using failure reports to discover model blind spots provides unique advantages over standard manual and algorithmic methods. Primarily, aggregated failure reports can be a more cost and time-efficient option for finding systematic failures by surfacing commonly represented errors, a task that often requires significant manual labor. By including a more diverse set of users, this crowd auditing method can also detect and describe failures that developers had not considered

due to their own blind spots and biases [25]. Additionally, if the process is applied to real-world domains instead of using crowd workers, the process can identify failures that are not present in the test set or that arise due to issues like dataset shift [41, 52]. In the rest of this paper, we explore the challenges for collecting and analyzing failure reports, and describe the design choices and technical solutions we took to address them.

4 FAILURE REPORT COLLECTION

We define *failure reports* as text submissions from end-users of AI systems describing a model's failure for a certain instance. While failure reports are conceptually similar to bug reports for software systems, their content differs substantially. ML systems do not provide any additional insight into why an error occurred, like stack traces and logs in software systems [67]. A failure report and data instance is the only information reported back to the developer, and thus a report's entire value comes from the text. Given the open-ended structure of failure reports, developers can make various design choices when collecting reports, including the question they pose to end-users and the response format. Here, we describe these design options and the specific choices we made for this work.

4.1 What is a Failure Report?

Failure reports are free-text responses from end-users describing certain features of an instance. While we tested other types of reports, specifically short 'tags' describing an error, we found that descriptive sentences were necessary to capture nuance like actions and complex image features. Free-text sentences are also a low barrier of entry for end-users that do not require any additional training, instruction, or domain knowledge. Beyond being a short sentence, there is considerable freedom in what developers can ask end-users to describe in their failure report. Through testing in different domains and results from our user study, we found that the type of report a developer should collect depends on a model's domain.

In complex tasks with ambiguous ground truth or performance measures, it is often most useful to collect responses of *how* a model is failing. For example, in a system that automatically generates captions for an image, collecting descriptions of how the captions are wrong (e.g., the dog is holding a frisbee although the caption says it is a tennis ball) can provide model developers with a mental map of the kinds of mistakes the model is making.

In more well-defined domains like classification, collecting responses of *why* a model is failing is often more useful since the type of error is apparent (the wrong class). For a system that detects if someone is wearing glasses, for example, it is often clear what the problem is (e.g., the glasses were not detected) but not why it happened (e.g., the glasses had thin frames, or the person was looking to the side). Descriptions of what aspects of an instance end-users think caused the algorithm to fail can provide the context needed to help developers find systematic errors.

In this work, we used different techniques for the two image domains we analyzed. In the eyeglass detection example, we collected *why* descriptions since the model failures are clear, and for image captioning we collected *how* descriptions to discover the ways in which captions were wrong.

4.2 Methods for Collecting Reports

In addition to *what* developers ask end-users to describe, developers also have to decide *how* they will collect reports. We define two primary methods for collecting failure reports: from end-users in a deployed setting or workers on a crowdsourcing platform. For a deployed AI system, failure reports can be collected in the same way software bug reports are - when users see that a model's output is wrong, they can submit free-response text reports through a reporting interface. This can be implemented as a dedicated error reporting UI, or integrated into the existing support

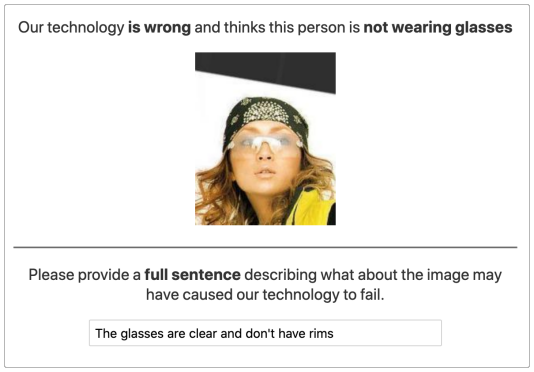

Fig. 3. The web-based interface used to collect failure reports. We provided Amazon Mechanical Turk (AMT) workers with a series of 3 images that a model had failed for. In this example, the AMT worker was given a false negative error from the eyeglass detection model which they attributed the error to the person's glasses being rimless and reflective. These reports were used in *Deblinder* for our user study and evaluation.

and feedback features of a product. Crowdsourcing platforms are a viable alternative that can be used to gather failure reports if end-users do not typically see a system's outputs or developers want immediate results. The primary difference between the two methods is that crowdsourcing platforms require the developer to have a set of labeled failure instances to show crowdworkers, while in deployed systems end-users discover the model failures themselves.

To run controlled experiments and reproduce our results, we used the crowdsourcing strategy in this work and created a web-based interface for end-users to describe how or why a model failed for an instance. We used Amazon Mechanical Turk (AMT) to collect failure reports using a web interface we created seen in Figure 3. We showed each worker three instances, and for each instance asked the worker for a description of either how or why, depending on the domain, the model had failed. The task took an average of 5 minutes, and we paid participants $1, the equivalent of $12 per hour. We used the system to collect failure reports for both the eyeglass detection and image captioning models, collecting 163 and 55 failure reports for each domain respectively. This report collection interface is a proof of concept for this work - real-world systems would implement their own report collection platforms and processes specific to their product.

Since AMT is generally representative of US internet users (with some small, consistent variation [26]), it provides a good approximation of the type and quality of reports that would be gathered in public-facing AI applications. To control AMT responses' quality, we limited participants to those who had an approval rating of over 98% with over 500 completed tasks. We found the responses to be of high quality generally, corroborated both by direct comments by study participants (AI developers) and the consistency and impact of the failures they discovered (Section 6). One developer commented that "there's no way this is Turker data" due to the reports' high quality, and all developers found consistent, evidenced blind spots from the AMT failure reports. Future work could explore methods for incentivizing good reports and study the differences between failures discovered by the general public and data scientists.

5 FAILURE REPORT ANALYSIS

Failure reports provide the core insights into model errors, but to discover valid systematic failures developers have to extract aggregate patterns and supporting evidence from hundreds or thousands of text reports. To support this process, we present the design and implementation of a visual

analytics system, *Deblinder*, that aggregates failure reports in a word embedding visualization and lets developers create and test their blind spot hypotheses.

There is a rich literature of visual analytics approaches for exploring large corpora of text [34]. This text analysis is often described as *sensemaking*, the process by which people organize large datasets to understand and validate patterns [44, 45, 63] The sensemaking framework specific to data analysis by Pirolli and Card [45] has been used to derive the design requirements for many visual analytics systems. One such system is Jigsaw, a visualization system for exploring large text datasets [22]. The authors used algorithmic methods to summarize, aggregate, and organize documents in a visual interface. Other systems using the sensemaking framework include the Aruvi system for analytical reasoning [57] and Apolo for exploring large networks [14].

Like these existing visual systems, we derive design challenges for *Deblinder* using the established sensemaking framework by Pirolli and Card [45]. Sensemaking consists of two processes - information foraging, where an analyst finds evidence and forms initial ideas, and synthesis, where the analyst creates and tests formal hypotheses. The three primary design challenges we define come directly from the central stages of the sensemaking process, the [R1] evidence file, [R2] schema, and [R3] hypotheses. We tailor the challenges specifically for text analysis and AI failures.

5.1 Design Requirements

A visual analytics system should address the following challenges for analyzing failure reports :

R1. Extract, filter, and search for high-level concepts.
Text-based failure reports should be summarized in a way that scales to thousands of reports. Developers should be able to quickly see an overview of high-level concepts derived from raw reports. Developers should also be able to apply their own domain knowledge of possible failures by filtering and searching for specific concepts and reports.

R2. Meaningfully organize concepts and failure reports.
To develop and validate hypotheses, it is useful to know what concepts and reports are most similar to each other. This context allows users to brainstorm hypotheses that include a variety of similar concepts and find reports that may fit into their existing hypotheses.

R3. Create, manage, and validate multiple hypotheses.
Developers should be able to create, track, and test formal hypotheses of systematic errors. They should be able to create and name hypotheses with supporting evidence, failure reports, and testing instances. When a developer has a hypothesis of a certain systematic failure, they should have methods for validating their hypothesis.

5.2 System Design

The *Deblinder* interface, seen in Figure 1, is primarily composed of two views, the Failure Report Embedding (Figure 1A) and the Hypothesis Panel (Figure 1C). The Failure Report Embedding is the first point of entry and provides an interactive, aggregated visualization of failure reports. Developers explore and select reports from this view to generate hypotheses in the Hypothesis Panel. In the Hypothesis Panel, they then create and collect evidence for hypothesized systematic failures. Developers can then use two different strategies to validate their hypotheses: modify instances to see the model's new output, or collect additional instances to see if the systematic failure generalizes to similar instances.

5.2.1 Making Sense of Failure Reports. The Failure Report Embedding is the primary interface with failure reports. The collected reports are free-response sentences semantically describing either how or why an end-user believes the AI system failed for a given instance. To give users a high-level view of which types of errors are happening most often, we extract *concepts* from

Fig. 4. The **Failure Report Embedding** view used to explore and derive insights from reports, shown with reports for the image captioning model. Concepts are extracted from reports and displayed using word embeddings. When a user hovers over a concept, it shows a few example reports and instances. Clicking on a concept expands all the reports in the reports panel, allowing users to add them to their hypothesis. Hovering over any instance in the interface shows the full instance in the upper left corner with the model's prediction.

the reports **[R1]**, key phrases from the reports representing commonly mentioned terms. These concepts are aggregated, counted, and displayed in the Failure Report Embedding visualization, an example of which can be seen in Figure 4 for the domain of eyeglass detection.

To extract concepts from the reports, we use RAKE (Rapid Automatic Keyword Extraction), a domain-agnostic keyphrase extraction algorithm [54]. The extracted high-level concepts are then shown in a two-dimensional word embedding visualization. To create the visual embedding, we use word vectors and dimensionality reduction. For each extracted concept, we calculate its embedding vector using the GloVe (Global Vectors for Word Representation) algorithm [43], which gives us a quantitative measure of how similar each concept is to each other. To project the 300-dimensional GloVe vectors of each concept into a two-dimensional plot, we use the UMAP dimensionality reduction algorithm, which maintains the locality of points in the reduced space [40].

We decided to use a visual embedding instead of a table or word cloud to fulfill the proposed design requirements. A central requirement for the failure reports visualization is showing the semantic similarity between reports **[R2]**. This is helpful because it can help users find similar concepts that may not share the same text. For example, in the eyeglass detection model, the concepts *frames* and *rims* are closely related and often interchangeable. A visual embedding groups similar concepts like these in space - in the same eyeglass domain, the concepts for *glasses*, *frames*, and *lenses* are located close to each other. In contrast, the concepts for *helmet* and *hat* are tightly coupled in another area of the embedding. This locality allows users to quickly see the relationship between concepts and explore reports that are similar in meaning. The visual embedding medium is also interactive and allows developers to quickly go from concepts to individual reports, which would not be easily accomplished with a table.

To improve the usability of the Failure Report Embedding we use empirical findings from visualization research. For each extracted concept, we calculate how many times it is mentioned across all failure reports, and set the size and opacity of its text accordingly. This both gives

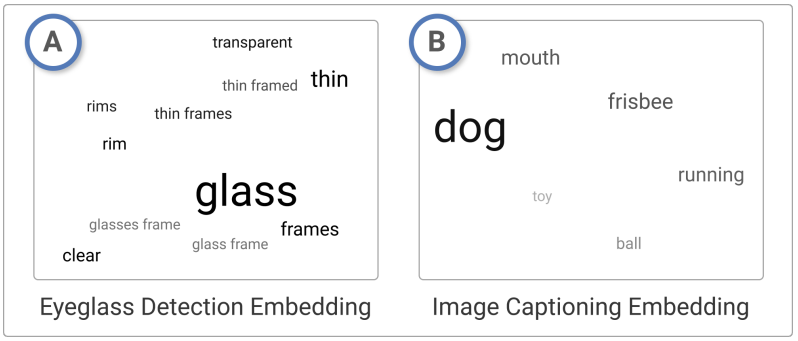

Fig. 5. Example areas of the Failure Report Embedding for the eyeglass detection and image captioning domains. In the eyeglass detection embedding, reports for *thin*, *transparent*, and *rims* are close to glasses, priming the hypothesis for clear or thin glasses frames. For image captioning we see the concepts for *frisbee* and *running*, hinting at the common error of captions mentioning a nonexistent frisbee for standing dogs.

developers an idea of what the most common reports are **[R2]** and helps deal with the scalability issue of too many reports crowding the embedding and reducing legibility. Additionally, research has found that keyword discovery, a task similar to exploring concepts, is best aided by changes to the spatial layout and font of the text [19]. The final usability feature we include is a form of semantic zooming [7]. As users zoom into sections of the Failure Report Embedding, we rescale the font size and opacity, spreading out the phrases and making smaller concepts easier to see.

After exploring high-level concepts, users next have to look at specific reports and instances to support their hypotheses. When a user hovers over a concept, a preview box appears around the phrase. It includes a few example instances, the number of reports with that concept, and excerpts from a few reports giving context to how end-users are using the concept. When the user clicks on the concept, it pins the phrase and expands all the related reports in the *reports panel* (Figure 1B). The reports panel shows all the instances and full failure reports for the selected concept and allows users to add the reports to their hypotheses.

Lastly, it is often the case that a developer has domain knowledge about the model they might want to apply. Towards this end, we include several utility features in *Deblinder* **[R1]**. We let users add new keywords to the Failure Report Embedding, which are placed with the appropriate scale and coloring where they semantically belong. There is also a filtering feature that dynamically highlights in the embedding any concept the developer writes. Users may also want to find key phrases or words in the failure reports themselves, which we enable with the search functionality in the reports panel. With these utility features, developers can correlate their intuitions and findings with the failure reports they gather.

5.2.2 Creating and Tracking Hypotheses. Once developers have generated initial hypotheses using the Failure Report Embedding, they can use the Hypothesis Panel (see Figure 1C) to create, track, and further test systematic failures. Developers can name and track multiple hypotheses consisting of representative failure reports, and use two validation methods to test their initial ideas.

The first step developers take is creating and adding evidence to a hypothesis, which is done in the Hypothesis Panel. At the top of the Hypothesis Panel developers can name the current hypothesis according to a specific type of error. Using the dropdown menu, they can switch between different hypotheses or create a new hypothesis **[R3]**. An example of the Hypothesis Panel can be seen in Figure 6 for the domain of eyeglass detection.

When a hypothesis is selected, the first section of the panel shows the failure reports the developer has added to the hypothesis. These reports are instances end-users reported due to the failure reason described by the hypothesis. When developers hover on each failure report , the concept's related reports are highlighted in the Failure Report Embedding, allowing the user to find potential new concepts to explore and expand their hypotheses **[R3]**.

5.2.3 Validating Hypotheses. While failure reports provide strong evidence of a model's systematic failures, they are not sufficient evidence to conclude that the issue generalizes. Failure reports are an incomplete source of evidence for two primary reasons. The first is that failure reports only show us instances for which the model is wrong. In practice, it may be the case that the model is correct for most instances described by a hypothesis. In the eyeglass detection domain, it may be that most people with clear glasses are correctly classified at a similar rate as people with dark-rimmed glasses, despite various failure reports mentioning clear glasses. The second way in which failure reports are insufficient evidence is in testing how plausible end-user reasons for model failure are. The reasons for failure reported by end-users are subjective and may not be real causes of a model's failure. Using the same example as above, it could be the case that the images of people with clear glasses were incorrectly classified because they were all worn low on the face, confounding the provided reason of clear glasses.

To address these issues, we introduce two methods in *Deblinder* that allow users to test their hypotheses and validate how viable each one is: additional instance search and image manipulation. The first testing method available in *Deblinder* is running the model on additional instances described by a hypothesis. By looking at the model's performance on additional instances, developers get a better approximation of how their model performs on a given blind spot compared to the overall dataset. While the most straightforward way to find additional examples would be to search through the training and testing sets, this method presents a couple of significant difficulties. First, finding images that match an arbitrary human-defined feature, for example, 'a person with clear glasses looking sideways', is a challenging and generally unsolved problem. Second, if the types of model failures reported by end-users are not present in the original dataset, additional instances of the reported issues would not be found with this strategy.

Due to these difficulties, we include a different method for discovering new instances inspired by Beat the Machine [4]: we allow developers to find additional data in the wild **[R3]**. Search engines are often able to find data for arbitrary descriptions, so *Deblinder* embeds a search feature for finding images using the Flickr API, which developers can use directly from the interface to find and upload images for a given search term. *Deblinder* also allows developers to upload new instances developers find using other means. The AI system is then run on the new instances to get the model's output, and developers can indicate whether the model is correct or incorrect. A percentage bar tracks the model's accuracy on the new instances, giving developers an approximation for how well their hypotheses generalize.

This technique can also be used for the last stage of the process, Model Improvement, by collecting additional training data. As we show in Section 8, the additional images collected as evidence can be added to the training set to improve the model's performance. To be model and framework agnostic, *Deblinder* does not have a feature for directly retraining the AI system but lets users export the discovered images to be included in future training sets.

The second validation technique available in *Deblinder* lets developers directly test their hypotheses by modifying an instance and looking at the model's changed output. For a reported or additional instance, a developer can download the instance, modify it, and re-upload it **[R3]**. Developers can use any tool or system they are comfortable with to modify the instance, ranging from photo manipulation software to algorithmic manipulations. This technique takes inspiration from

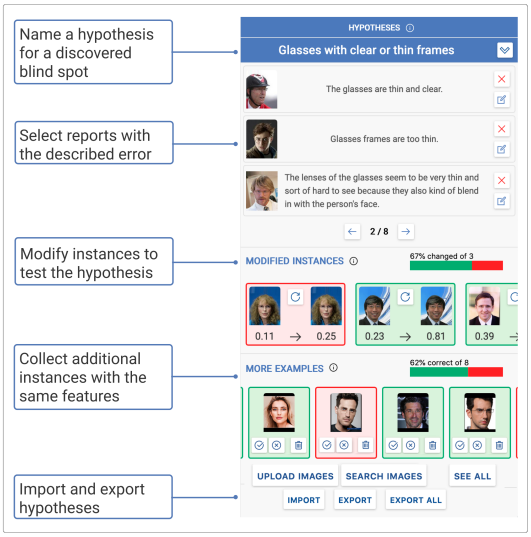

Fig. 6. The **Hypothesis Panel** developers use to create, track, and test hypotheses. Developers can name hypotheses according to the features describing a systematic failure. For each hypothesis, developers add reports that match the hypothesis reason. They can then test the hypotheses with the modified instance and additional example strategies. Lastly, users can import and export hypotheses files for collaboration and integration into existing ML development tools.

Cabrera et al. [12]'s web-based interface that allows users to remove features of an image and see the model's new output. However, we decided not to include any predefined image manipulations to allow developers to use techniques they are familiar with to modify the image in any way. For the example hypothesis above of clear glasses, the developer may download an image of someone with clear glasses and darken the glasses frames. They can then upload the edited image and see the model's output. The model output for both the original and modified instance are presented to the developer, who indicates whether the model's output changed as expected. As with the additional image validation, a percentage bar tracks how many of the modified instances changed as expected, giving the user a heuristic for how valid their hypothesis is.

5.3 Implementation

Deblinder is a web-based system built using Svelte[1] and D3[2] for the front-end, and Flask[3] for the server, both of which can be run locally. The Flask server hosts the machine learning model, which is used to get the model output for additional examples and modified images. The resulting hypotheses and evidence can be saved, exported, and shared. *Deblinder* will be open source and can be adapted to new domains by updating the instance preview.

6 USER STUDY METHODOLOGY

The goal of the user study was twofold: to understand the real-world applications and limitations of failure reports, and to evaluate *Deblinder*'s usability for discovering systematic failures. To derive these insights, we conducted semi-structured interviews and think-aloud studies with 10

[1] https://svelte.dev/
[2] https://d3js.org/
[3] https://flask.palletsprojects.com/en/1.1.x/

participants. The 10 participants were recruited through university and company email lists. They all had at least three years of ML experience and either published an ML research project or deployed a production AI system. 8 were Ph.D. candidates in a computer science-related discipline (machine learning, human-computer interaction, or computer science), and 2 were developers at different software companies who work with deployed models. The study lasted up to one hour, and participants were compensated with a $20 Amazon gift card.

All studies were done remotely over video conferencing and consisted of three primary sections. The first part of the study was a semi-structured interview to understand the types of AI systems participants work with and their current process for discovering and fixing systematic failures. The second part of the study aimed to evaluate the usability and workflow of *Deblinder* by having participants use the system to create and validate hypotheses. After a ten-minute introduction and walk-through of the system's major features, we tasked participants with thinking aloud while creating and testing two or three failure hypotheses. We then provided them with additional images to upload into *Deblinder* and try one of the hypothesis testing strategies. In the final section, we discussed with participants the potential applications and limitations of using failure reports in their own models and systems.

Participants used a web-hosted version of *Deblinder* loaded with the reports collected for the domain of eyeglass detection described in Sections 3.1 and 4.2. We focused on eyeglass detection since facial recognition systems are widespread and have received attention for real-world issues [10].

7 USER STUDY RESULTS

Two researchers independently analyzed the results of the user study using thematic analysis and affinity diagramming. Additionally, the hypotheses participants generated using *Deblinder* were aggregated in Table 1. We found that with 10 participants the themes for the applications of failure reports, usability of *Deblinder*, and failures participants discovered (Table 1) converged significantly.

7.1 *Deblinder* and the Sensemaking Process

The crowd auditing workflow. All participants found the workflow of *Deblinder* to follow their mental process for discovering and testing systematic errors. One participant who conducts education research described their model debugging process specifically in terms of sensemaking: "... we're going to get together and talk and say like, what are our hypotheses about what's really going on here? [...] And then we test the hypothesis by saying: Okay. Here's what we think might be going on. Can we test whether that seems to be true? If we look at thousands of students rather than just this one instance."

Other participants described similar debugging processes in their own work and identified benefits specific to a crowd auditing process with failure reports. One participant who works at a self-driving car company stated that "This is basically the exact workflow at some level that we kind of do, but it's more of a manual process and it's usually after the fact instead of before the fact." Another participant stated that they "become the crowd and try to simulate the test set" when validating their models and concluded that our process "is great for democratizing AI."

Exploring failure reports with the report embedding. Eight of the developers came up with their hypotheses from looking at the Failure Report Embedding, while two had preexisting ideas of potential errors they either looked for in the embedding or used the search functionality to discover. While some participants noticed that the embedding contained extraneous concepts, for example, *glasses*, they generally found the embedding useful, with one participant mentioning it was "very good at finding patterns."

Table 1. **Participant Hypotheses.** The hypotheses participants generated in the study using *Deblinder*. We joined similar hypotheses together and show how many participants reported each one.

Hypothesis	# of Participants
Thin, clear, or no rims	7
Dark or tinted lenses	5
Eyes occluded	4
Bad image quality	2
Looking sideways	2
Eyebrows confused with frames	1
Shadow over eyes	1
Oddly positioned glasses	1

While all participants eventually found the failure reports useful and understood what they represented, it took two participants a few interactions to fully understand them. These participants were initially confused about both where the reports came from and what they represented. One participant thought that the reports were hypotheses that other developers had created, while another did not know the reports were only for wrongly classified images. We do not believe this is a severe flaw in the *Deblinder* system or crowd auditing process since, in a deployed setting, the developer would be more involved in defining and collecting the failure reports. The developer would then have a clear mental model of what the raw data being visualized is.

Creating and validating failure hypotheses. All participants found the Hypothesis Panel interface useful for tracking and testing hypotheses. There was also significant overlap in the systematic failures participants described, as seen in Table 1, suggesting that *Deblinder* can surface the most prevalent patterns present in a set of failure reports.

An interesting dynamic we discovered during the studies was three participants' uncertainty with how specific to make their hypotheses. One participant was unsure whether to create a hypothesis for "face is obstructed", or the more specific hypothesis of "hair is covering their face". Another participant overcame this uncertainty by beginning with a more general hypothesis and then refining it if, through testing, the general hypothesis was not confirmed. We intentionally let the developer choose the granularity of hypotheses since, depending on the domain, certain description levels may be more helpful for fixing the discovered issues.

Another interaction four participants wished for was richer interactions between hypotheses. In *Deblinder* each hypothesis has its own isolated sets of reports and instances. Participants at times wished they could drag and drop additional instances into other hypotheses or reuse some of their collected reports. Richer interactions between hypotheses could create a more seamless experience.

All participants found the additional instance validation useful for testing their hypotheses. The modified instance interaction also reflected participants' expectations. For example, before being introduced to the system, one participant mentioned that "counterfactuals seem like a logical choice" for testing their model.

7.2 Applications and Limitations of Failure Reports

Suitable domains. Through conversations with our participants, we found a pattern for which AI systems would be best suited for crowd auditing using failure reports. Our participants worked with various models, data types, and domains in both industry and research. We found that the primary

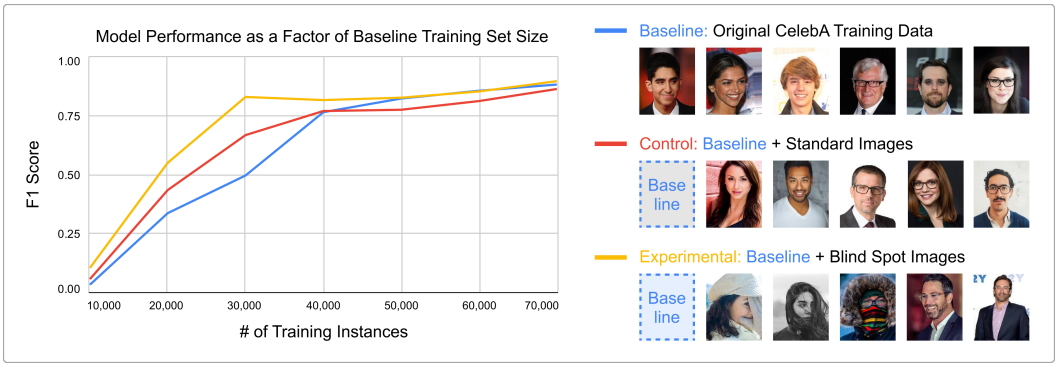

Fig. 7. Retraining the eyeglass detection model with data from the blind spots identified with *Deblinder* shows significant improvement in performance over retraining with randomly selected additional data.

factor dictating the usefulness of *Deblinder* was how human-understandable the model's input and output are. For an end-user to generate a probable explanation, they have to understand what the data and output represent. For example, financial data or signal processing will generally not work well, as it is often difficult for humans to spot potential issues or edge cases. This is not the case for domains like pedestrian detection and image captioning, where it is often evident to humans when a model is wrong and what the problem is. Many AI systems focus on these human-understandable domains for which end-users can provide viable reports, and over half of our participants worked with such systems. Failure reports can even be used in more complex domains if domain experts are seeing the model output and can describe potential failures, for example, radiologists and x-rays.

Model errors in practice. Nine participants had encountered situations in which their models had systematic errors in practice. For example, one participant developed a system for tracking hands using a video feed and found that the model tended to fail when a hand was at the edge of the screen. After some investigation, they attributed this issue to a "center bias" in the data, which was mostly made up of hands in the middle of the video or picture frame. Following this example, most participants attributed their system's systematic failures to the data rather than the model.

One of our participants works on a production AI system with multiple chained models. In this case, it is hard to pinpoint the exact source of the problem, i.e., which model is at fault for the failure, using the testing mechanisms in *Deblinder*. Despite this, failure reports can still be used to characterize failures the overall system is making.

8 EXPERIMENTAL VALIDITY OF FAILURE REPORTS

While the user study showed that developers can find and validate various types of systematic failures using *Deblinder*, there is an open question if these insights can be used to improve model performance. To directly test the final stage of our crowd auditing process, Model Improvement, we gathered additional data from the hypotheses participants identified in the user study and retrained the glasses detection model. We hypothesized that adding data specifically from systematic failure groups will improve a model more than adding the same type of data. If this hypothesis holds, it suggests images from the blind spots developers found were underrepresented in the data, leading to the failures crowd auditing was able to detect with failure reports and *Deblinder*.

We collected additional data by using a script to scrape Google Images, limited to permissibly licensed images. To find particular images, we used search terms like 'person wearing glasses' and 'person with covered face'. We then downloaded the images for each search term and removed

those that did not fit the description. This data gathering process can also be done using the similar image search functionality in *Deblinder*

To isolate the impact of blind spot images on model performance, we used three different conditions. The baseline condition was a random subset of the original CelebA training data, which has 160,000 headshots of celebrities [38]. The control condition consisted of the images from the baseline condition plus 644 additional images: 354 images of people with glasses and 290 of people without glasses. The final experimental condition also consisted of the baseline images plus a different set of 673 additional images: 361 images of people with glasses (145 with clear glasses and 215 with covered or occluded faces and glasses) and 312 images of people without glasses and covered or occluded faces. These additional images represent the most common blind spots discovered by participants in the user study, as seen in Table 1. For both the control and experimental condition, we originally collected 750 images each but removed the images that did not fit the search terms.

To measure the impact of additional images on model performance, we retrained every model at different-sized subsets of baseline training images. When training the baseline model, 600 random images from the training set were added to control for the additional data in the other conditions. The results can be seen in Figure 7, with the F1 score used to measure performance on the test set since it gives a more accurate view of performance for imbalanced data. We find that at almost every training size, the experimental condition performs better than both the control and baseline conditions, confirming our hypothesis that collecting data from blind spots can better improve model performance.

There are several additional results from this experiment. First, the control condition performed consistently better than the baseline condition. We believe that this is likely due to the new images differing from the training data in dimensions like image quality and face positioning that help the model generalize, a phenomenon that has been shown in ImageNet models [52]. Second, we find that the impact of additional images on performance diminishes as the training set size increases. This is likely due to the number of additional images (~600) being two orders of magnitude smaller than the original training set (40,000-70,000), minimizing the impact of additional images on training.

9 DISCUSSION

9.1 Generalization of Failure Reports and Deblinder

Our implementation of *Deblinder* represents one example design of how to make failure reports useful for discovering systematic failures. Notably, *Deblinder* is focused on the setting of one analyst looking at various failure reports in the domain of images. Future work could explore methods for using crowdsourcing to synthesize and make sense of reports, further scaling the manual process enabled by *Deblinder*, or developing algorithmic techniques for extracting insights. This work also focused on text-based failure reports as the primary medium of reporting. More advanced techniques like data tagging could lead to better methods for clustering similar instances and extracting systematic failures.

The current version of *Deblinder* generalizes to image domains beyond binary classification. We show this by collecting and visualizing failure reports for an image captioning model on pictures of animals. We used a pre-trained PyTorch captioning model[4] and a subset of images from the Flickr30K dataset [46]. In this case, how the model failed is not obvious to end-users, as it might fail to recognize the objects in the image correctly or fail to generate syntactical text. For this reason, we collected failure reports for *how* the model failed instead of why in contrast to the eyeglass

[4]https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning

detection example. We used the same interface and Amazon Mechanical Turk process as described in Section 4.2.

The resulting Failure Report Embedding for image captioning can be seen in Figure 5B, and demonstrates some interesting insights. These include *frisbee* being a common concept; as it turns out, the captioning model describes many standing dogs as holding a frisbee regardless of whether they are holding one or not. *Running* is also prevalent, as the model describes animals as 'running' if they are just standing or upright. While *Deblinder* was designed to work with any image domain, it is also extensible to include other domains like video, text, and tabular data.

9.2 Limitations and Future Work

Failure reports provide a valuable snapshot into AI systems' real-world systematic errors, but model blind spots are complex with various dimensions. Extensions to *Deblinder* and future work could further developers' ability to understand and improve their model's real-world performance.

Fixing systematic failures. *Deblinder*'s similar image search can be used to collect more training data and improve model performance, but it is not a complete solution. Future work could explore more effective methods for fixing the detected failures. Algorithmic techniques like clustering and classification could supplement similar image search and be used to label more training data [30, 37]. For natural images, this includes pre-trained models for image classification and object detection like those for OpenImages [35]. Even without more data, recent data programming methods like slice-based learning can be used to improve model performance in areas of high error [16].

Better testing and tracking. While developers appreciated the hypothesis testing techniques, they require nontrivial developer effort and only provide approximations to the ground truth validity of hypotheses. To scale the testing of hypotheses, crowd-based systems similar to Beat the Machine could be used to task crowd workers with editing or discovering new images [4]. Research into scalable and novel testing techniques could also help improve hypothesis evaluation. These include more advanced algorithmic methods for finding additional instances or solutions like Generative Adversarial Nets (GANs) that could create synthetic examples to test a blind spot hypothesis [47]. Additional validation methods would also help counter confirmation bias that might come from confounders or biases in individual sources of evidence.

As developers begin to change and improve AI systems, it is important to track their performance over time. Is the model doing better for some regions of error? Is it regressing for others? Are there new blind spots? Tracking an AI system's performance *Deblinder* could be extended to visualize the evolution of failures over time, tracking whether data and model changes have their desired effect.

Complex hypotheses and domain knowledge. The *Deblinder* system and the overall process generally matched developers' model debugging processes, but we discovered some limitations and opportunities for improvement in our user study. *Deblinder* was designed to work with discrete, separable blind spots, but we found that they are often much more ambiguously defined. Some participants wanted to be able to nest and group hypotheses, like, for example, putting the 'hat covering face' hypothesis in the more general hypothesis for 'face is obstructed'. This ambiguity could impact later stages of the process like what type of data is collected. Future work could explore more complex methods for visualizing and organizing hypotheses and supporting evidence.

Another interaction that participants desired was a more direct way to include their domain knowledge and insights into the Failure Report Embedding and system. Richer interactions with the embedding, such as allowing developers to add their own tags and descriptions to failure reports, could deepen the insights developers can get from the reports.

Better understanding failure reports. We have shown for two distinct domains that end-users provide probable failure reports, and experimentally showed for one of the domains that those

findings can be used to improve model performance. While developers can verify blind spots in *Deblinder*, future work could explore human descriptions of model errors further, for example, understanding the types of failures people are more likely to detect. A more complex understanding of the human side of crowd auditing could help developers decide when to deploy the process and guide the development of techniques and systems for encouraging better reporting.

10 CONCLUSION

AI systems are being deployed to a growing number of societally impactful domains. To improve their performance and understand their potential impacts, developers must know what types of errors their models are making. We introduce crowdsourced *failure reports*, descriptions of how or why a model failed, and show how they can be used to detect and validate systematic failures in AI systems. To synthesize hundreds or thousands of text reports, we design and implement a visual analytics system, *Deblinder*, for aggregating, visualizing, and validating insights from failure reports. Tightening the loop between model development and real-world evaluation is essential for developing safe and responsible AI systems, and insights from end-users provide a rich new data source for augmenting this process.

ACKNOWLEDGMENTS

This material is based upon work supported by an Amazon grant, a Block Center for Technology and Society grant, a National Science Foundation grant under No. IIS-2040942, and the National Science Foundation Graduate Research Fellowship Program under grant No. DGE-1745016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Amazon, the Block Center or the National Science Foundation.

REFERENCES

[1] Yongsu Ahn and Yu Ru Lin. 2020. Fairsight: Visual analytics for fairness in decision making. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 1086–1095. https://doi.org/10.1109/TVCG.2019.2934262

[2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019* (2019), 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[3] Paul André, Aniket Kittur, and Steven P. Dow. 2014. Crowd synthesis: Extracting categories and clusters from complex data. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW* (2014), 989–998. https://doi.org/10.1145/2531602.2531653

[4] Josh M Attenberg, Pagagiotis G Ipeirotis, and Foster Provost. 2011. Beat the machine: Challenging workers to find the unknown unknowns. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*.

[5] Gagan Bansal and Daniel S. Weld. 2018. A coverage-based utility model for identifying unknown unknowns. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018), 1463–1470.

[6] Solon Barocas and Andrew D Selbst. 2018. Big Data's Disparate Impact. *SSRN Electronic Journal* 671 (2018), 671–732. https://doi.org/10.2139/ssrn.2477899

[7] B. B. Bederson, J. Meyer, and L. Good. 2000. Jazz: An extensible zoomable user interface graphics toolkit in Java. *UIST (User Interface Software and Technology): Proceedings of the ACM Symposium* (2000), 171–180. https://doi.org/10.1016/b978-155860915-0/50016-0

[8] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What Makes a Good Bug Report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Atlanta, Georgia) *(SIGSOFT '08/FSE-16)*. Association for Computing Machinery, New York, NY, USA, 308–318. https://doi.org/10.1145/1453101.1453146

[9] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. [n.d.]. Duplicate bug reports considered harmful... really?. In *2008 IEEE International Conference on Software Maintenance*. IEEE, 337–345.

[10] Joy Buolamwini and Timnit Gebru. 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*. 77–91.

[11] 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 2019 - Proceedings*. 46–56. https://doi.org/10.1109/VAST47406.2019.8986948

[12] Ángel Alexander Cabrera, Fred Hohman, Jason Lin, and Duen Horng Chau. 2018. Interactive Classification for Deep Learning Interpretation. (2018), 1–5. http://arxiv.org/abs/1806.05660

[13] Joseph Chee Chang, Aniket Kittur, and Nathan Hahn. 2016. Alloy: Clustering with crowds and computation. *Conference on Human Factors in Computing Systems - Proceedings* (2016), 3180–3191. https://doi.org/10.1145/2858036.2858411

[14] Duen Horng Chau, Aniket Kittur, Jason I. Hong, and Christos Faloutsos. 2011. Apolo: Making sense of large network data by combining rich user interaction and machine learning. *Conference on Human Factors in Computing Systems - Proceedings* (2011), 167–176. https://doi.org/10.1145/1978942.1978967

[15] Nan Chen Chen, Jina Suh, Johan Verwey, Gonzalo Ramos, Steven Drucker, and Patrice Simard. 2018. Anchorviz: Facilitating classifier error discovery through interactive semantic data exploration. *International Conference on Intelligent User Interfaces, Proceedings IUI* (2018), 269–280. https://doi.org/10.1145/3172944.3172950

[16] Vincent S. Chen, Sen Wu, Zhenzhen Weng, Alexander Ratner, and Christopher Ré. 2019. Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices. NeurIPS (2019). http://arxiv.org/abs/1909.06349

[17] Justin Cheng and Michael S. Bernstein. 2015. Flock: Hybrid crowd-machine learning classifiers. *CSCW 2015 - Proceedings of the 2015 ACM International Conference on Computer-Supported Cooperative Work and Social Computing* (2015), 600–611. https://doi.org/10.1145/2675133.2675214

[18] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. 2019. Slice finder: Automated data slicing for model validation. *Proceedings - International Conference on Data Engineering* 2019-April (2019), 1550–1553. https://doi.org/10.1109/ICDE.2019.00139

[19] Cristian Felix, Steven Franconeri, and Enrico Bertini. 2018. Taking Word Clouds Apart: An Empirical Investigation of the Design Space for Keyword Summaries. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 657–666. https://doi.org/10.1109/TVCG.2017.2746018

[20] Kristie Fisher, Scott Counts, and Aniket Kittur. 2012. Distributed sensemaking: Improving sensemaking by leveraging the efforts of previous users. *Conference on Human Factors in Computing Systems - Proceedings* (2012), 247–256. https://doi.org/10.1145/2207676.2207711

[21] Eureka Foong, Darren Gergle, and Elizabeth M. Gerber. 2017. Novice and expert sensemaking of crowdsourced feedback. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 1–18. https://doi.org/10.1145/3134680

[22] Carsten Görg, Zhicheng Liu, Jaeyeon Kihm, Jaegul Choo, Haesun Park, and John Stasko. 2013. Combining computational analyses and interactive visualization for document exploration and sensemaking in jigsaw. *IEEE Transactions on Visualization and Computer Graphics* 19, 10 (2013), 1646–1663. https://doi.org/10.1109/TVCG.2012.324

[23] Nitesh Goyal. 2015. Designing for Collaborative Sensemaking: Using Expert & Non-Expert Crowd. (2015). http://arxiv.org/abs/1511.06053

[24] Geon Heo, Yuji Roh, Seonghyeon Hwang, Dayun Lee, and Steven Euijong Whang. 2020. Inspector Gadget: A Data Programming-Based Labeling System for Industrial Images. *Proc. VLDB Endow.* 14, 1 (Sept. 2020), 28–36. https://doi.org/10.14778/3421424.3421429

[25] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé, Miroslav Dudík, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need? *Conference on Human Factors in Computing Systems - Proceedings* (2019), 1–16. https://doi.org/10.1145/3290605.3300830

[26] Panos Ipeirotis. [n.d.]. Demographics of mechanical Turk CeDER-10-01.pdf. ([n. d.]). http://archive.nyu.edu/fda/bitstream/2451/29585/2/CeDER-10-01.pdf

[27] He Jiang, Xiaochen Li, Zhilei Ren, Jifeng Xuan, and Zhi Jin. 2018. Toward Better Summarizing Bug Reports With Crowdsourcing Elicited Attributes. *IEEE Transactions on Reliability* PP (2018), 1–21. https://doi.org/10.1109/TR.2018.2873427

[28] Shubhra Goyal Jindal, Student Member, and Arvinder Kaur. 2020. Automatic Keyword and Sentence-Based Text Summarization for Software Bug Reports. *IEEE Access* 8 (2020), 65352–65370. https://doi.org/10.1109/ACCESS.2020.2985222

[29] Minsuk Kahng, Dezhi Fang, and Duen Horng Chau. 2016. Visual exploration of machine learning results using data cube analysis. *HILDA 2016 - Proceedings of the Workshop on Human-In-the-Loop Data Analytics* (2016). https://doi.org/10.1145/2939502.2939503

[30] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. 2018. Interpretability beyond feature attribution: Quantitative Testing with Concept Activation Vectors (TCAV). *35th International Conference on Machine Learning, ICML 2018* 6 (2018), 4186–4195.

[31] Miryung Kim, Thomas Zimmermann, Robert Deline, and Andrew Begel. 2018. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1024–1038. https://doi.org/10.1109/TSE.2017.2754374

[32] Aniket Kittur, Andrew M. Peters, Abdigani Diriye, and Michael R. Bove. 2014. Standing on the schemas of giants: Socially augmented information foraging. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW* (2014), 999–1010. https://doi.org/10.1145/2531602.2531644

[33] Aniket Kittur, Andrew M. Peters, Abdigani Diriye, Trupti Telang, and Michael R. Bove. 2013. Costs and benefits of structured information foraging. *Conference on Human Factors in Computing Systems - Proceedings* (2013), 2989–2998. https://doi.org/10.1145/2470654.2481415

[34] Kostiantyn Kucher and Andreas Kerren. 2015. Text visualization techniques: Taxonomy, visual survey, and community insights. *IEEE Pacific Visualization Symposium* 2015-July (2015), 117–121. https://doi.org/10.1109/PACIFICVIS.2015.7156366

[35] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. 2020. The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale. *International Journal of Computer Vision* 128, 7 (2020), 1956–1981. https://doi.org/10.1007/s11263-020-01316-z

[36] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Eric Horvitz. 2017. Identifying unknown unknowns in the open world: Representations and policies for guided exploration. *31st AAAI Conference on Artificial Intelligence, AAAI 2017* Settles 2009 (2017), 2124–2132.

[37] Anthony Liu, Santiago Guerra, Isaac Fung, Gabriel Matute, Ece Kamar, and Walter Lasecki. 2020. Towards Hybrid Human-AI Workflows for Unknown Unknown Detection. *The Web Conference 2020 - Proceedings of the World Wide Web Conference, WWW 2020* 2020 (2020), 2432–2442. https://doi.org/10.1145/3366423.3380306

[38] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[39] Travis Mandel, Jahnu Best, Randall H. Tanaka, Hiram Temple, Chansen Haili, Sebastian J. Carter, Kayla Schlechtinger, and Roy Szeto. 2020. Using the Crowd to Prevent Harmful AI Behavior. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW2 (2020). https://doi.org/10.1145/3415168

[40] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *The Journal of Open Source Software* 3, 29 (2018), 861.

[41] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recognition* 45, 1 (2012), 521–530. https://doi.org/10.1016/j.patcog.2011.06.019

[42] Besmira Nushi, Ece Kamar, and Eric Horvitz. 2018. Towards Accountable AI: Hybrid Human-Machine Analyses for Characterizing System Failure. *HCOMP* (2018). http://arxiv.org/abs/1809.07424

[43] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[44] Peter Pirolli and Stuart Card. 1999. Information foraging. *Psychological Review* 106, 4 (1999), 643–675. https://doi.org/10.1037/0033-295X.106.4.643

[45] Peter Pirolli and Stuart Card. 2005. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. *Proceedings of International Conference on Intelligence Analysis* 2005, January (2005), 2–4. https://doi.org/10.1007/s13398-014-0173-7.2 arXiv:9809069v1 [gr-qc]

[46] Bryan A. Plummer, Liwei Wang, Chris M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. 2017. Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models. *International Journal of Computer Vision* 123, 1 (2017), 74–93. https://doi.org/10.1007/s11263-016-0965-7

[47] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), 1–16.

[48] Iyad Rahwan, Manuel Cebrian, Nick Obradovich, Josh Bongard, Jean François Bonnefon, Cynthia Breazeal, Jacob W. Crandall, Nicholas A. Christakis, Iain D. Couzin, Matthew O. Jackson, Nicholas R. Jennings, Ece Kamar, Isabel M. Kloumann, Hugo Larochelle, David Lazer, Richard McElreath, Alan Mislove, David C. Parkes, Alex 'Sandy' Pentland, Margaret E. Roberts, Azim Shariff, Joshua B. Tenenbaum, and Michael Wellman. 2019. Machine behaviour. *Nature* 568, 7753 (2019), 477–486. https://doi.org/10.1038/s41586-019-1138-y

[49] Ramya Ramakrishnan, Ece Kamar, Besmira Nushi, Debadeepta Dey, Julie Shah, and Eric Horvitz. 2019. Overcoming Blind Spots in the Real World: Leveraging Complementary Abilities for Joint Execution. *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), 6137–6145. https://doi.org/10.1609/aaai.v33i01.33016137

[50] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. 2014. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering* 40, 4 (2014), 366–380. https://doi.org/10.1109/TSE.2013.2297712

[51] Alexander Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data Programming: Creating Large Training Sets, Quickly. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 3574–3582.

[52] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. Do ImageNet classifiers generalize to ImageNet? *36th International Conference on Machine Learning, ICML 2019* 2019-June (2019), 9413–9424.
[53] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D. Williams. 2017. Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 61–70. https://doi.org/10.1109/TVCG.2016.2598828
[54] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic Keyword Extraction from Individual Documents. *Text Mining: Applications and Theory* March (2010), 1–20. https://doi.org/10.1002/9780470689646.ch1
[55] Adrian Schröter, Nicolas Bettenburg, and Rahul Premraj. 2010. Do stack traces help developers fix bugs?. In *Proceedings - International Conference on Software Engineering*. 118–121. https://doi.org/10.1109/MSR.2010.5463280
[56] Andrew D. Selbst, Danah Boyd, Sorelle A. Friedler, Suresh Venkatasubramanian, and Janet Vertesi. 2019. Fairness and abstraction in sociotechnical systems. *FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency* (2019), 59–68. https://doi.org/10.1145/3287560.3287598
[57] Yedendra B. Shrinivasan and Jarke J. Van Wijk. 2008. Supporting the analytical reasoning process in information visualization. *Conference on Human Factors in Computing Systems - Proceedings* (2008), 1237–1246. https://doi.org/10.1145/1357054.1357247
[58] Tom Simonite. 2018. When it comes to gorillas, google photos remains blind. *Wired, January* 13 (2018).
[59] Chengnian Sun, David Lo, Siau Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings* (2011), 253–262. https://doi.org/10.1109/ASE.2011.6100061
[60] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. *Proceedings - International Conference on Software Engineering* 1 (2010), 45–54. https://doi.org/10.1145/1806799.1806811
[61] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. *SSRN Electronic Journal* (2017), 1–52. https://doi.org/10.2139/ssrn.3063289
[62] Daisuke Wakabayashi. 2018. Self-driving Uber car kills pedestrian in Arizona, where robots roam. *The New York Times* 3 (2018), 19.
[63] Karl E Weick. 1995. Sensemaking in Organizations (Foundations for Organizational Science). *Star* (1995).
[64] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. {E}rrudite: Scalable, Reproducible, and Testable Error Analysis. *Proceedings of the 57th Conference of the Association for Computational Linguistics* (2019), 747–763. https://www.aclweb.org/anthology/P19-1073
[65] Shamima Yeasmin, Chanchal K. Roy, and Kevin A. Schneider. 2014. Interactive visualization of bug reports using topic evolution and extractive summaries. *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014* (2014), 421–425. https://doi.org/10.1109/ICSME.2014.66
[66] Jie Zhang, Xiao Yin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. 2015. A survey on bug-report analysis. *Science China Information Sciences* 58, 2 (2015), 1–24. https://doi.org/10.1007/s11432-014-5241-2
[67] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. 2010. What makes a good bug report? *IEEE Transactions on Software Engineering* 36, 5 (2010), 618–643. https://doi.org/10.1109/TSE.2010.63

Received January 2021; revised April 2021; accepted July 2021