On Human Intellect and Machine Failures: Troubleshooting Integrative Machine Learning Systems

Besmira Nushi,[1] **Ece Kamar,**[2] **Eric Horvitz,**[2] **Donald Kossmann**[1,2]

[1]ETH Zurich, Department of Computer Science, Switzerland
[2]Microsoft Research, Redmond, WA, USA

Abstract

We study the problem of troubleshooting machine learning systems that rely on analytical pipelines of distinct components. Understanding and fixing errors that arise in such integrative systems is difficult as failures can occur at multiple points in the execution workflow. Moreover, errors can propagate, become amplified or be suppressed, making blame assignment difficult. We propose a human-in-the-loop methodology which leverages human intellect for troubleshooting system failures. The approach simulates potential component fixes through human computation tasks and measures the expected improvements in the holistic behavior of the system. The method provides guidance to designers about how they can best improve the system. We demonstrate the effectiveness of the approach on an automated image captioning system that has been pressed into real-world use.

Introduction

Advances in machine learning have enabled the design of integrative systems that perform sophisticated tasks via the execution of analytical pipelines of components. Despite the widespread adoption of such systems, current applications lack the ability to understand, diagnose, and fix their own mistakes which consequently reduces users' trust and limits future improvements. Therefore, the problem of understanding and troubleshooting failures of machine learning systems is of particular interest in the community (Sculley et al. 2015). Our work studies *component-based* machine learning systems composed of specialized components that are individually trained for solving specific problems and work altogether for solving a single complex task. We analyze how the special characteristics of these integrated learning systems, including *continuous* (non-binary) success measures, *entangled* component design, and *non-monotonic* error propagation, make it challenging to assign blame to individual components. These challenges hinder future system improvements as designers lack an understanding of how different potential fixes on components may improve the overall system output.

Approach. We introduce a troubleshooting methodology which relies on crowdworkers to identify and fix mistakes

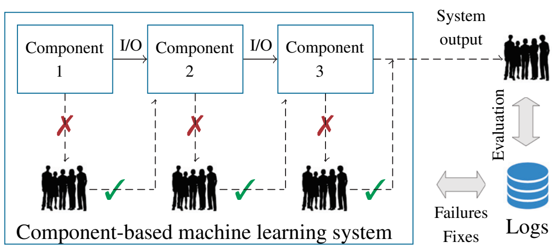
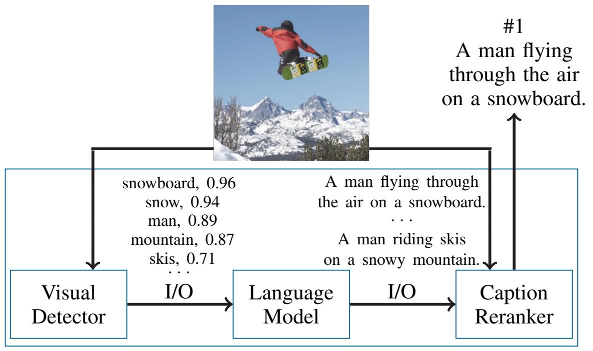


Figure 1: Troubleshooting with humans in the loop

in existing systems. Human intervention is crucial to the approach as human fixes simulate improved component output that cannot be produced otherwise without significant system development efforts. Figure 1 shows the main flow of our approach. First, workers evaluate the system output without any fix to analyze the current system state. To simulate a component fix, the input and output of the component accompanied with the fix description are sent to a crowdsourcing platform as microtasks. Once workers apply the targeted fixes for a component, the fixed output is integrated back into the running system, which thereby generates an improved version of the component. The system is executed as a simulation with the injected fix and the output is evaluated again via crowdworkers. The overall process collects a valuable set of log data on system failures, human fixes, and their impact on the final output. This data can then be analyzed to identify the most effective combination of component improvements to guide future development efforts.

Case study. We apply our methodology to a state-of-the-art integrative learning system developed to automatically caption images (Fang et al. 2015). The system involves three machine learning components in a pipeline, including *visual detection*, *language generation*, and *caption ranking*. The multimodal nature of this case study allows us to demonstrate the applicability of the approach for components processing different forms of data and carrying out various tasks. The methodology reveals new insights on the error dynamics previously unknown to the designers, and offers recommendations on how to best improve the system. For example, in contrast to what system designers had assumed, improving the Reranker is more effective than improving the Visual Detector. Experiments highlight the benefits of mak-



#1
A man flying through the air on a snowboard.
snowboard, 0.96
snow, 0.94
man, 0.89
mountain, 0.87
skis, 0.71
...
A man flying through the air on a snowboard.
...
A man riding skis on a snowy mountain.
Visual Detector
I/O
Language Model
I/O
Caption Reranker

Figure 2: The image captioning system

ing informed decisions about component fixes as their effectiveness varies greatly (18%, 3% and 27% for the three components respectively).

Background and Problem Characterization

We now define the problem of troubleshooting component-based machine learning systems. We start by describing the image captioning system as a running example and then continue with the problem formalization.

Case Study: An Image Captioning System

The system (Fang et al. 2015) that we use as a case study automatically generates captions as textual descriptions for images. This task has emerged as a challenge problem[1] in artificial intelligence as it involves visual and language understanding and has multiple real-world applications, including the provision of descriptions for assisting visually impaired people. Figure 2 shows the system architecture, consisting of three machine learning components. The first and third components leverage convolutional neural networks combined with multiple instance learning (Zhang, Platt, and Viola 2005). The second one is a maximum-entropy language model (Berger, Pietra, and Pietra 1996).

Visual Detector. The first component takes an image as an input and detects a list of words associated with recognition scores. The detector recognizes only a restricted vocabulary of the 1000 most common words in the training captions.

Language Model. This component is a statistical model that generates likely word sequences as captions based on the words recognized from the Visual Detector, without having access to the input image. The set of the 500 most likely image captions and the respective log-likelihood scores are forwarded to the Caption Reranker.

Caption Reranker. The task of the component is to rerank the captions generated from the Language Model and select the best match for the image. The Reranker uses multiple features among which the similarity between the vector representations of images and captions. The caption with the highest ranking score is selected as the final best caption.

Dataset. All components are individually trained on the MSCOCO dataset (Lin et al. 2014) which was built as an image captioning training set and benchmark. It contains 160,000 images as well as five human-generated captions for all images. We use images randomly sampled from the validation dataset to evaluate our approach.

[1]http://mscoco.org/dataset/#captions-challenge2015

Problem Characterization

Problem context. This work studies machine learning systems consisting of several components designed to carry out specific tasks in the system. The system takes a set of data as *system input* and the individual components work together to produce a final *system output*. We assume that the system architecture is provided to the methodology by system designers by specifying:

1. The set of system components along with the *component input* and *output* data types.
2. A set of directed communication dependencies between components denoting the input / output exchange. The whole set of dependencies defines the system *execution workflow*. In our methodology, we only handle acyclic dependencies but allow for branching.

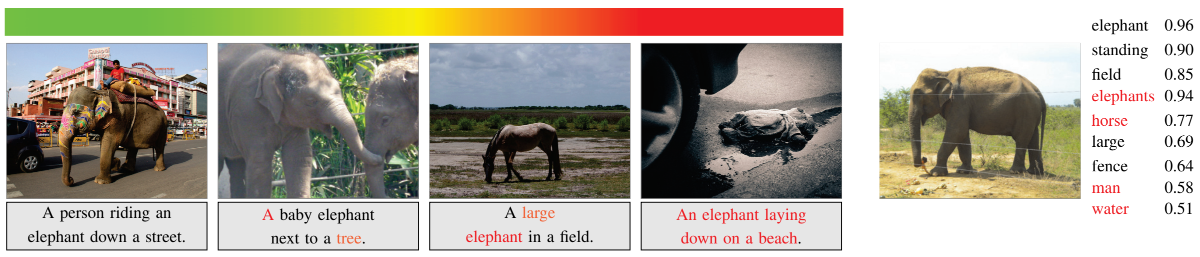
Problem definition. Troubleshooting of component-based machine learning systems can be decoupled to answering the following two questions:

Question 1: *How does the system fail?*
The system designer is interested in identifying and measuring the different types of *system failures* and their frequencies as well as the failures of individual components in the system context.

Question 2: *How to improve the system?*
System failures can be addressed by various potential fixes applicable to individual components. To guide future efforts on improving the system, the system designer is interested in knowing the effects of component fixes on: (i) specific input instances, and (ii) the overall system output quality. The first requirement is relevant to fine-grained instance-based debugging that a troubleshooting methodology should implement, while the second one provides guidance on future efforts to improve the whole system.

The first question aims at analyzing the current state of the system. The second question explores future opportunities and investigates the efficiency of different strategies to improve the system. Next, we examine the special characteristics of component-based machine learning systems that make the problem of troubleshooting challenging. These characteristics differentiate this problem from previous work on troubleshooting and motivate our methodology.

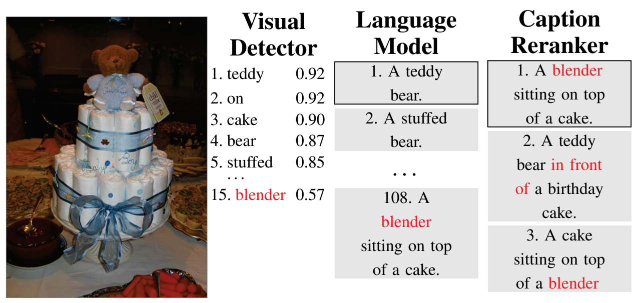
Continuous quality measures. Uncertainty is inherent in machine learning components. When these components work together to solve complex tasks, the measure of quality for individual components and the system as a whole is no longer binary, rather it spans a wide spectrum. Therefore, the evaluation of these systems needs to go beyond accuracy metrics to deeper analysis of system behavior. For instance, Figure 3 shows a few examples from image captioning where the system and component output varies in quality and the types of mistakes. Troubleshooting in this


A person riding an elephant down a street.
A baby elephant next to a tree.
A large elephant in a field.
An elephant laying down on a beach.

elephant	0.96
standing	0.90
field	0.85
elephants	0.94
horse	0.77
large	0.69
fence	0.64
man	0.58
water	0.51

(a) System output (b) Visual Detector output

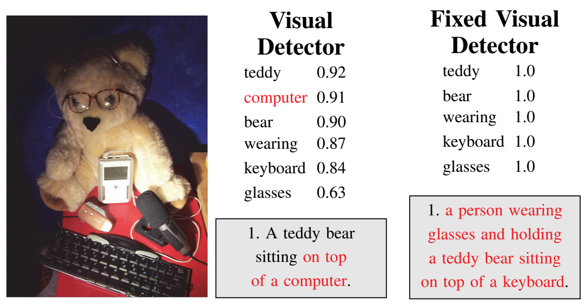
Figure 3: Continuous output quality in the captioning system



Visual Detector	
1. teddy	0.92
2. on	0.92
3. cake	0.90
4. bear	0.87
5. stuffed	0.85
…	
15. blender	0.57

Language Model	Caption Reranker
1. A teddy bear.	1. A blender sitting on top of a cake.
2. A stuffed bear.	2. A teddy bear in front of a birthday cake.
…	3. A cake sitting on top of a blender
108. A blender sitting on top of a cake.	

Figure 4: Component entanglement in the captioning system



Visual Detector		Fixed Visual Detector	
teddy	0.92	teddy	1.0
computer	0.91	bear	1.0
bear	0.90	wearing	1.0
wearing	0.87	keyboard	1.0
keyboard	0.84	glasses	1.0
glasses	0.63		
1. A teddy bear sitting on top of a computer.		1. a person wearing glasses and holding a teddy bear sitting on top of a keyboard.	

Figure 5: Non-monotonic error in the captioning system

quality continuum where all components are only partially correct is non-trivial.

Complex component entanglement. In component-based machine learning systems, components have complex influences on each other as they may be tightly coupled or the boundaries between their responsibilities may not be clear. When the quality of a component depends on the output of previous components, blame cannot be assigned to individual components without decoupling imperfection problems in component inputs. Figure 4 illustrates a typical scenario of component entanglement in the image captioning system. The final caption is clearly unsatisfactory as it mentions a non-existing object (`blender`). However, the Visual Detector assigns a low score to the word `blender` (0.57), which makes the detector only partially responsible for the mistake. The Language Model is also partially responsible as it creates a sentence with low commonsense awareness. Finally, the caption reranker chooses as the best caption a sentence that includes a word with a low score. In this example, errors from all components are interleaved and it is not possible to disentangle their individual impact on the final error.

Non-monotonic error. We note that improving the outputs of components does not guarantee system improvement. On the contrary, doing so may lead to quality deterioration. For example, when components are tuned to suppress erroneous behavior of preceding components, applying fixes to the earlier ones may result to unknown failures. Figure 5 shows an example of non-monotonic error behavior. Here, the Visual Detector makes a mistake by including `computer` in the list. The initial assumption would be that if the list is fixed so that it contains only the prominent words, then the quality of the caption should increase. In reality, the caption after the fix is more erroneous than the original. Since the language model finds a teddy bear wearing glasses unlikely, it creates a caption that mentions a `person` instead.

Human-in-the-loop Methodology

Due to these problem characteristics, blame assignment is challenging in integrative systems and analyzing only the current state of the system is not sufficient to develop strategies for system improvement. As shown in Figure 1, our methodology overcomes these challenges by introducing humans in the loop for: (i) simulating component fixes, and (ii) evaluating the system before and after fixes to directly measure the effect of future system improvements.

Methodology setup. The troubleshooting methodology is applicable to systems that follow the assumptions of: (i) *system modularity* with clearly defined component inputs and outputs, and (ii) *human interpretability* of the component input / output. To apply the methodology to a new system, the system designer provides the set of components and their input/outputs within the system execution workflow. After identifying a list of component fixes that can potentially improve the system, the system designer formulates corresponding crowdsourcing tasks for these fixes and the overall system evaluation. Both types of tasks should describe the high-level goal of the system, the context in which it operates as well as its requirements (*e.g.* an image captioning system designed to assist users with visual impairments). In addition, component fixing tasks should be appropriately formulated so that their expected output matches the output of implementable fixes that the system designer plans to test.

Troubleshooting steps. The execution of the methodology is guided by the *fix workflow*, which is a combination of various component fixes to be evaluated. The system designer chooses which fix workflows to execute and evaluate for the purpose of troubleshooting. For a given fix workflow, the steps of our methodology are as follows:

1. *Current system evaluation* — workers assess the final output of the current system on various quality measures.
2. *Component fix simulation* — for each fix in the workflow, workers complete the respective micro-task for examining and correcting the component output.
3. *Fix workflow execution* — executing a fix workflow involves integrating the corrected outputs of each component into the system execution.
4. *After-fix system evaluation* — workers re-evaluate the new system output after incorporating component fixes.

When a workflow includes fixes for multiple components, steps 2 and 3 need to be repeated so that the fixes of earlier components are reflected on the inputs of later components.

Troubleshooting outcomes. Applying human fix workflows simulates improved component states and helps system designers to observe the effect of component fixes on system performance, overcoming the challenges raised by the problem characteristics.

1. *Continuous quality measures* — Comparing the system quality before and after various fix workflow executions not only can quantify the current quality of system and component output, but it can also isolate and quantify the effect of individual component fixes. For example, if many components are partially failing and are possibly responsible for a specific error, the system designer can test the respective fixes, systematically understand their impact, and decide which are the most promising ones.
2. *Non-monotonic error* — Non-monotonic error propagation can be disclosed when the overall system quality drops after a component fix. When such a behavior is observed, the system designer can conclude that although these fixes may improve the internal component state, they are not advisable to be implemented in the current state of the system as they produce negative artifacts in the holistic system.
3. *Complex component entanglement* — Entanglement detection requires the execution of workflows with different combinations of fixes to measure the individual and the joint effect of component fixes. For example, if two consecutive components are entangled, individual fixes in either one of them may not improve the final output. However, if both components are fixed jointly, this may trigger a significant improvement. The designer could also use this information to detect entanglement and potentially correct the system architecture in future versions.

Troubleshooting the Image Captioning System

We now describe the customized crowdsourcing tasks for our case study for both system evaluation and component fixes. Table 1 lists all component fixes specifically designed for this case study. The task design is an iterative process in collaboration with system designers so that human fixes can appropriately simulate implementable improvements.

Component	Fix description
Visual Detector	Add / remove objects
Visual Detector	Add / remove activities
Language Model	Remove noncommonsense captions
Language Model	Remove non-fluent captions
Caption Reranker	Rerank Top 10 captions

Table 1: Summary of fixes for the image captioning system.

System evaluation. The system evaluation task is designed to measure different quality metrics associated with captions as well as the overall *human satisfaction*. The task shows workers an image-caption pair and asks them to evaluate the following quality measures: *accuracy* (1-5 Likert scale), *detail* (1-5 Likert scale), *language* (1-5 Likert scale), *commonsense* (0-1), and *general evaluation*. For each measure, we provided a detailed description along with representative examples. However, we intentionally did not instruct workers for the general evaluation to prevent biasing them on which quality measure is more important (*e.g.* accuracy vs. detail).

Visual Detector fixes. We designed two different tasks for fixing object and activity detections respectively represented by nouns and verbs in the word list. The *object fix* shows workers the input image with the list of nouns present in the visual detector output. Workers are asked to correct the list by either removing objects or adding new ones. The *activity fix* has a similar design. We intentionally group *addition* and *removal* fixes to simulate improvements of the Visual Detector in precision and recall as potential implementable fixes in this component. The result of any Visual Detector fix is a new word list which is passed to the Language Model together with the worker agreement scores (*e.g.* majority vote).

Language Model fixes. These fixes are designed for removing sentences that are either not commonsense or not fluent. The tasks do not share the input image with the worker, as the Language Model itself does not have access to the image. In the *commonsense fix*, workers mark whether a caption describes a likely situation that makes sense in the real world. For example, the caption `A cat playing a video game` has no commonsense awareness in a general context. In the *language fix*, the goal is to evaluate the language fluency of captions in a 1-5 Likert scale. In addition, workers highlight problematic parts of the sentence which they think would make the caption fluent if fixed appropriately. The resulting list of problematic segments is a reusable resource to filter out captions that contain the same patterns.

Both fixes simulate improved versions of the Language Model that generate commonsense sentences in a fluent language. To integrate Language Model fixes in the system execution we exclude the noncommonsense and the non-fluent captions from the list forwarded to the Caption Reranker.

Caption Reranker fixes. This task shows an image together with the corresponding top 10 captions from the Reranker (in random order), and asks workers to pick up to 3 captions that they think fit the image best. The answers are then aggregated via majority vote and the caption with the highest agreement is selected as the new system output.

	`Eval.`	`Sat.`	`Unsat.`
Accuracy (1-5)	3.674	4.474	2.579
Detail (1-5)	3.563	4.265	2.601
Language (1-5)	4.509	4.693	4.256
Commonsense (0-1)	0.957	1.000	0.898
General (1-5)	3.517	4.306	2.437
%Satisfactory (0-1)	57.8%	100%	0%

Table 2: Current system evaluation.

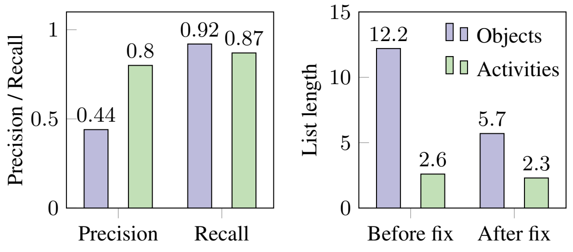


Figure 6: Visual Detector fixes and the component state.

Experimental Evaluation

The evaluation of the captioning system with our methodology uses an `Evaluation` dataset of 1000 images randomly selected from the MSCOCO validation dataset. All experiments were performed on Amazon Mechanical Turk. We report the system quality based on human assessments. An additional analysis using automatic machine translation scores can be found in the longer version of the paper (Nushi et al. 2016).

Current System State

First, we evaluate the current system state as shown in Table 2. To gain a deeper understanding of the system performance, we divide the `Evaluation` data in two datasets: `Satisfactory` and `Unsatisfactory` based on the general evaluation score collected from the system evaluation task. We consider every answer in 1-3 as an unsatisfactory evaluation, and every other answer in 4-5 as satisfactory. All instances whose original caption reaches a majority agreement on being satisfactory belong to the `Satisfactory` dataset. The rest is classified as `Unsatisfactory`.

Result: Only 57.8% of the images in the `Evaluation` dataset have a satisfactory caption. The comparison between the `Satisfactory` and `Unsatisfactory` partitions shows that the highest discrepancies happen for the *accuracy* and *detail* measures, highlighting the correlation of accuracy and detail with the overall satisfaction.

Current Component State

An additional functionality of a human-assisted troubleshooting methodology is to evaluate the quality of the existing individual system components.

Visual Detector Figure 6 shows the precision and recall of the Visual Detector for both objects and activities when compared to the human-curated lists created from the majority vote aggregation of multiple workers' fixes. In the same figure, we also show the size of the lists before and after applying human fixes.

Result: The Visual Detector produces longer lists for objects than for activities but with lower precision and recall.

	Commonsense fix	Language fix	Both fixes
Top1-`Eval.`	8.0%	22.9%	**25.0%**
Top10-`Eval.`	8.6%	21.7%	**27.1%**
Top1-`Val.`	3.0%	15.2%	**16.1%**
Top10-`Val.`	2.6%	14.2%	**14.9%**

Table 3: Percentage of pruned captions from the Language Model and the component state.

	No fix	Object	Activity	Addition	Removal	All fixes
Accuracy	3.674	**4.045**	3.681	3.709	4.000	4.035
Detail	3.563	3.900	3.590	3.604	3.880	**3.916**
Language	**4.509**	4.505	4.427	4.521	4.423	4.432
Csense.	0.957	0.947	0.940	**0.957**	0.933	0.942
General	3.517	**3.848**	3.510	3.549	3.796	3.831
%Sat.	57.8%	**69.1%**	57.1%	58.5%	66.8%	68.0%

Table 4: Visual Detector fixes — `Evaluation` dataset.

Language Model. In the Language Model fixes, we examine only those captions from the Language Model that are among the Top10 best captions in the Caption Reranker. Given that many of the 500 generated sentences never appear as best captions of the Reranker, and the Language Model output is quite extensive to be fully fixed via crowdsourcing, we focus only on those captions that are likely to impact the system output. Table 3 shows the percentage of captions pruned after applying the two fixes. The `Validation` dataset here represents the whole MSCOCO dataset which contains 40,504 images.

Result: Due to self-repetition within the dataset, fixes generated for the 1000 images of the `Evaluation` dataset generalize well to the whole `Validation` set, pruning 16.1% of the Top1 captions and 14.9% of the Top10 captions. Language fixes have a higher coverage than the commonsense ones.

Caption Reranker. Caption Reranker fixes also focus only on the Top10 best captions. After reranking this set with the crowdsourcing majority vote, we observe that the best caption changes for 76.9% of the images. In 46.1% of the cases, the original caption was never chosen by any of the workers. For 19.2% of the images, the majority of workers reported that they could not find any caption in the list that is a good fit for the image. These cases are indeed more serious failures that cannot be recovered through reranking only.

Component Fixes and System Improvement

Visual Detector fixes. Table 4 shows results from applying the four types of fixes on the Visual Detector. These fixes increase the number of satisfactory captions in the dataset up to 17.6% compared to the initial state of the system. Ob-

	No fix	Commonsense	Language	All fixes
Accuracy	3.674	3.698	3.696	**3.712**
Detail	3.563	3.583	3.590	**3.602**
Language	4.509	4.575	4.618	**4.632**
Csense.	0.957	0.973	0.974	**0.982**
General	3.517	3.546	3.557	**3.572**
%Sat.	57.8%	58.5%	59.2%	**59.3%**

Table 5: Language Model fixes — `Evaluation` dataset.

	No fix	Reranking (All fixes)
Accuracy	3.674	**4.145**
Detail	3.563	**3.966**
Language	4.509	**4.626**
Csense.	0.957	**0.988**
General	3.517	**3.973**
%Sat.	57.8%	**73.6%**

Table 6: Caption Reranker fixes — `Evaluation` dataset.

ject fixes are more effective than the activity ones for two reasons. First, the precision of the Visual Detector is originally significantly lower for objects than for activities (0.44 vs. 0.8), which offers more room for improvement for object fixes. Second, activity fixes are limited by the shortcomings of the Language Model. Even when a corrected list of activities is provided to the Language Model, it may fail to form commonsense captions containing the corrected activities (*e.g.* `A woman holding an office`) due to non-monotonic error behavior of the system.

Result: The entangled design between the Visual Detector and the Language Model causes non-monotonic error propagation in particular for activity fixes.

Language Model fixes. Language fixes are generally more effective than the commonsense fixes as they have a higher coverage and they generalize better to other images. Fixes in the Language Model increase the number of satisfactory captions by only 3%.

Result: The impact of Language Model fixes is limited due to the fact that most captions with language mistakes also have other problems which cannot be fixed only through this component.

Caption Reranker fixes. As a final component, fixes in the Caption Reranker directly affect the final caption. This means that if there is a plausible caption in the Top10 set better than the original best caption, that caption is going to be ranked higher after the fix and will directly improve the system output. This explains why Caption Reranker improvements are higher than all other component fixes.

Result: The system improves by a factor of 27% after the Reranker fixes. Although this provides the most effective system improvement, its influence is limited to instances with at least one satisfactory caption in Top10, which is the case only for 80.8% of our dataset.

Complete fix workflow. Table 7 shows the improvements from each component and the complete fix workflow which sequentially applies all component fixes. Figure 7 decouples the results for the `Satisfactory` and `Unsatisfactory` partitions of the data set.

	No fix	Visual Detector	Language Model	Caption Reranker	All fixes
Accuracy	3.674	4.035	3.712	4.145	**4.451**
Detail	3.563	3.916	3.602	3.966	**4.247**
Language	4.509	4.432	4.632	4.626	**4.660**
Csense.	0.957	0.942	0.982	0.988	**0.998**
General	3.517	3.831	3.572	3.973	**4.264**
%Sat.	57.8%	68.0%	59.3%	73.6%	**86.9%**

Table 7: Complete fix workflow — `Evaluation` dataset

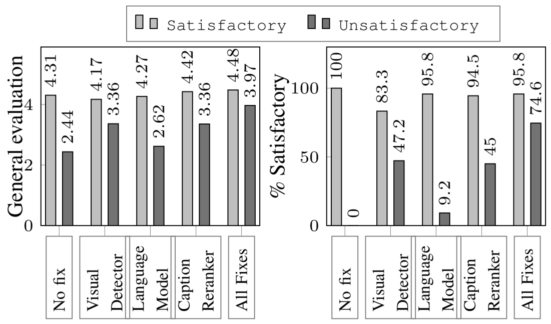


Figure 7: Human evaluation scores on the `Satisfactory` and `Unsatisfactory` datasets.

Result: The complete fix workflow increases the number of satisfactory captions by 50%. In contrast to the initial assumptions of system designers, fixes in the Caption Reranker are most effective due to the entanglement in the previous components. Most improvements come from the `Unsatisfactory` partition of the dataset. However, because of non-monotonic error behavior in specific instances, some of the fixes result in slight deteriorations on the `Satisfactory` partition (*e.g.* Visual Detector fixes).

Examples of Fix Integration

Figure 8 presents examples of different ways fix workflows affect the system output. Figure 8(a) is an example of fixes to the Visual Detector resulting in a satisfactory system output. In this example, workers removed the erroneous object `kite` and added `umbrella` which propagated the improvement to the final caption. In the larger dataset, successful propagations of individual component fixes to the final output are also observed for activity fixes, commonsense fixes, and caption refinement fixes. Figure 8(b) shows an example of fixes having a limited improvement on the final caption due to the commonsense barrier of the Language Model. In this example, the word `horse` was present in both the original and the fixed word list. However, none of the sentences generated by the Language Model could depict the situation in the image as it was not found to be likely. This example is not unique, the `Unsatisfactory` dataset contains a few more images of the same nature which describe an unlikely situation that are (at the moment) hard to be described by a statistical automated system. Figure 8(c) is an example in which improvements from fixes are hin-

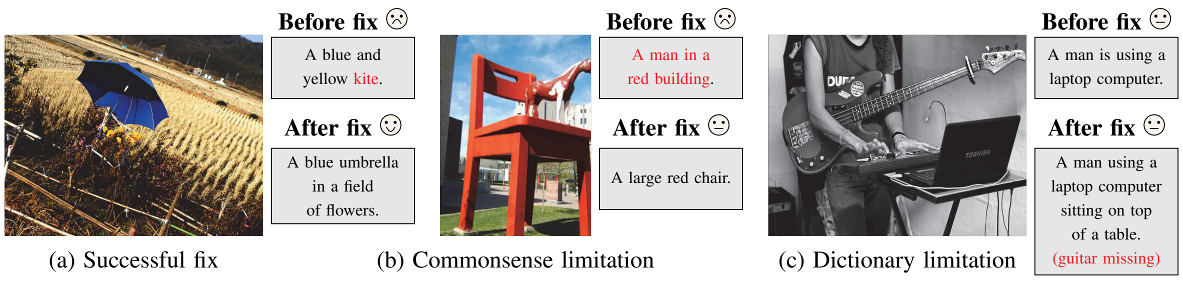

Before fix
A blue and yellow kite.
After fix
A blue umbrella in a field of flowers.
(a) Successful fix
Before fix
A man in a red building.
After fix
A large red chair.
(b) Commonsense limitation
Before fix
A man is using a laptop computer.
After fix
A man using a laptop computer sitting on top of a table. (guitar missing)
(c) Dictionary limitation

Figure 8: Examples of applying the complete fix workflow

dered by the limited size of the dictionary. Since the word `guitar` is not in the dictionary, the final caption fails to provide a satisfactory description.

Discussion

Quality control and methodology cost. For all crowdsourcing experiments we applied the following quality control techniques: (i) spam detection based on worker disagreement, (ii) worker training via representative examples and detailed instructions, (iii) periodical batching to prevent worker overload and keep them engaged. These techniques ensured high-quality data and enabled us to rely on the majority vote aggregation. Depending on the human computation task, specialized label aggregation techniques can also be leveraged to further improve data quality.

The cost of human-in-the loop troubleshooting depends on the number of components, the number of fixes, the fix workload, and the size of the dataset to be investigated. Our analysis covered various fix workflows on all components in the 1000 images `Evaluation` dataset which showed to be a good representative of the `Validation` dataset. The total cost of the complete fix workflow (the most expensive one) was $1,850, respectively spent in system evaluation ($250), Visual Detector fixes ($450), Language Model fixes ($900), and Caption Reranker fixes ($250). For a more specialized troubleshooting, the system designer can guide the process towards components that are prime candidates for improvement or on errors to which users are most sensitive. We provide further details on quality control and cost aspects in the longer version of the paper (Nushi et al. 2016).

Our analysis shows that even with a reasonably small subset of data, our human-in-the loop methodology can efficiently characterize system failure and identify potential component fixes. Alternative improvement methods (*i.e.* retraining) require a larger amount of data and oftentimes do not ensure significant improvements. This can happen due inherent learning barriers in the system or slow learning curves of underlying algorithms.

System improvement. The results from the methodology provide guidance on next steps to improve the captioning system. First, improving the Reranker emerges as the most promising direction to pursue. Second, due to entanglement issues, improvements on the Visual Detector are suppressed by the shortcomings of the Language Model. Therefore, Visual Detector fixes need to be accompanied with a more capable and commonsense Language Model. Note these insights cannot be revealed via other methodologies that do not involve human computation as it is challenging to automatically simulate improved components without significant engineering effort.

There are multiple ways how human input collected from simulating component fixes can help with permanently implementing component fixes. Human fixes on the Visual Detector reveal that the majority of mistakes are false detections. This issue can be addressed by improving model precision. Moreover, the data collected from language and commonsense fixes can be used for training better language models. or for immediately filtering out phrases and sentences flagged by workers. Finally, since a common type of reranking error occurs when the component chooses sentences with words scored low by the Detector, the Reranker can be improved by increasing the weight of the image-caption similarity score.

Generalizability. The general methodology we presented can be applied to a broad range of component-based systems that are designed to be modular and their component input / output is human-interpretable. Even in systems in which these assumptions do not hold in the functional component design, a new structure can be discovered by logically separating components in boundaries where data dependencies are guaranteed and the exchanged data can be analyzed by humans.

Applying the methodology to a new system requires the designer to customize the methodology by identifying component fixes, defining system quality measures and designing human computation tasks for evaluation and for simulating component fixes. In addition to the captioning system, we conceptually applied our methodology to two other systems: (i) question answering with knowledge bases and web search (Yih et al. 2015), and (ii) an email-based reminder for a personal assistant. Our feasibility analysis showed that both applications are compatible with the methodology and highlighted that the most crucial aspect of customization is providing careful and non-ambiguous training to crowdsourcing workers tailored to the system context. Given the novelty of the proposed human interventions, the resulting crowdsourcing tasks are expected to be different from the typical labeling tasks frequently encountered in today's platforms. Such problems are usually more interesting and engaging to crowdsourcing workers.

Alternative use cases. In this work, we discuss the benefits of using a human-in-the-loop methodology for troubleshooting integrative systems by simulating fixes within the existing individual components of a system. The methodology can additionally be used for further troubleshooting use cases as follows:

- *Component prototyping* — An interesting application of human interventions is to completely simulate the output of a component if building it is currently difficult or too expensive. This would allow system designers to make feasibility studies before developing a new component.
- *Architectural fixes* — Our goal in this work was to study single-component fixes that precisely follow the current given architecture. Besides individual component fixes, a system designer may be interested to explore fixes with architectural modifications (*e.g.* exposing the image to the Language Model), which is useful if designers are not bound to the initial architecture. Some examples include merging, dividing, or introducing new components. In these cases, the task design for human fixes needs to be adjusted according to the new architectural specification and the data exchange flow between components.
- *Imperfect fixes* — In our evaluation, we analyzed fixes that simulate perfect component states. However, building flawless components is not always feasible. Therefore, it is realistic to leverage the methodology to test imperfect fixes by only partially incorporating the human-generated corrections.

Related Work

Human input for Machine Learning. The contribution of crowdsourcing to machine learning has been mostly limited to the creation of offline data sets for learning (*e.g.*, (Lin et al. 2014; Sheng, Provost, and Ipeirotis 2008)), with recent interest in active crowd participation to the development of machine learning algorithms (Cheng and Bernstein 2015; Zou, Chaudhuri, and Kalai 2015; Chang, Kittur, and Hahn 2016). However, there has been only limited work on understanding and diagnosing errors of such systems. On debugging a single classifier, researchers have developed techniques for a domain expert to interact with the machine learning process (Chakarov et al. 2016; Kulesza et al. 2010; Patel et al. 2010; Attenberg, Ipeirotis, and Provost 2011). Our work contributes to this line of literature by studying the diagnosis of component-based systems, rather than individual predictive components, by leveraging the crowd input.

Crowdsourcing for Image Captioning. Crowdsourcing is heavily used for collecting data for object recognition and image captioning (Lin et al. 2014; Fang et al. 2015). In terms of evaluating the performance of a component-based system, previous work explored replacing different components with human input to measure the effect of component imperfections on system performance (Parikh and Zitnick 2011b; 2011a; Yao et al. 2015). This approach provides information on the current system but does not offer guidance on system improvements. Our work differs from these studies in that it evaluates the effect of different component fixes on system performance to guide decisions on future improvements.

Troubleshooting and Blame Assignment. The problems of error diagnosis and blame assignment have been studied for systems whose components are not machine learned and the state of components is binary through rule-based and model-based diagnosis approaches (Darwiche 2000). Breese and Heckerman developed Bayesian networks for predicting the state of each component and for making decisions about the next repair action to take (Breese and Heckerman 1996). In recent work, Sculley et. al., overviewed the challenges of maintaining and improving real-world machine learning systems highlighting error diagnosis as a critical task in particular for component-based systems (Sculley et al. 2015). An alternative way of improving machine learning algorithms is active learning (Settles 2010). Current techniques are applicable to single components (*e.g.* classifiers) but not to integrative systems with multiple components which is the focus of this work.

Conclusion and Future Work

We reviewed our efforts for troubleshooting component-based machine learning systems. The proposed methodology highlights the benefits of deeper integration of crowd input on troubleshooting and improving these integrative systems. Future work directions include the exploration of models that learn from the log data of our methodology to predict which fixes are most likely to improve the system quality for a given input. Such models can enable algorithms to query human fixes during system execution for improved system output. Finally, there is an opportunity to develop generalizable pipelines for automating human-in-the-loop troubleshooting of machine learning systems with reusable crowdsourcing task templates. Such a pipeline would provide valuable insights on system development and create a feedback loop in support of continuous improvement.

Acknowledgements

This research was initiated during an internship of Besmira Nushi at Microsoft Research. The work was supported in part by the Swiss National Science Foundation. The authors would like to thank Xiadong He, Margaret Mitchell, and Larry Zitnick for their support in understanding and modifying the execution of the image captioning system, as well as Paul Koch for his help while building the necessary project infrastructure. Our thanks extend to Dan Bohus for the valuable insights in current problems in integrative systems and Jaime Teevan for helpful discussions on this work.

References

Attenberg, J.; Ipeirotis, P. G.; and Provost, F. J. 2011. Beat the machine: Challenging workers to find the unknown unknowns. *Human Computation* 11:11.

Berger, A. L.; Pietra, V. J. D.; and Pietra, S. A. D. 1996. A maximum entropy approach to natural language processing. *Computational linguistics* 22(1):39–71.

Breese, J. S., and Heckerman, D. 1996. Decision-theoretic troubleshooting: A framework for repair and experiment. In *UAI*.

Chakarov, A.; Nori, A.; Rajamani, S.; Sen, S.; and Vijaykeerthy, D. 2016. Debugging machine learning tasks. *arXiv preprint arXiv:1603.07292*.

Chang, J. C.; Kittur, A.; and Hahn, N. 2016. Alloy: Clustering with crowds and computation. In *CHI*.

Cheng, J., and Bernstein, M. S. 2015. Flock: Hybrid crowd-machine learning classifiers. In *CHI*, 600–611. ACM.

Darwiche, A. 2000. Model-based diagnosis under real-world constraints. *AI magazine* 21(2):57.

Fang, H.; Gupta, S.; Iandola, F.; Srivastava, R. K.; Deng, L.; Dollár, P.; Gao, J.; He, X.; Mitchell, M.; Platt, J. C.; et al. 2015. From captions to visual concepts and back. In *CVPR*, 1473–1482.

Kulesza, T.; Stumpf, S.; Burnett, M.; Wong, W.-K.; Riche, Y.; Moore, T.; Oberst, I.; Shinsel, A.; and McIntosh, K. 2010. Explanatory debugging: Supporting end-user debugging of machine-learned programs. In *VL/HCC*, 41–48. IEEE.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *ECCV 2014*. Springer. 740–755.

Nushi, B.; Kamar, E.; Horvitz, E.; and Kossmann, D. 2016. On human intellect and machine failures: Troubleshooting integrative machine learning systems. *arXiv preprint arXiv:1611.08309*.

Parikh, D., and Zitnick, C. 2011a. Human-debugging of machines. *NIPS WCSSWC* 2:7.

Parikh, D., and Zitnick, C. L. 2011b. Finding the weakest link in person detectors. In *CVPR*, 1425–1432. IEEE.

Patel, K.; Bancroft, N.; Drucker, S. M.; Fogarty, J.; Ko, A. J.; and Landay, J. 2010. Gestalt: integrated support for implementation and analysis in machine learning. In *UIST*, 37–46. ACM.

Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.-F.; and Dennison, D. 2015. Hidden technical debt in machine learning systems. In *NIPS*.

Settles, B. 2010. Active learning literature survey. *University of Wisconsin, Madison* 52(55-66):11.

Sheng, V. S.; Provost, F.; and Ipeirotis, P. G. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD*, 614–622. ACM.

Yao, L.; Ballas, N.; Cho, K.; Smith, J. R.; and Bengio, Y. 2015. Trainable performance upper bounds for image and video captioning. *arXiv preprint arXiv:1511.04590*.

Yih, W.-t.; Chang, M.-W.; He, X.; and Gao, J. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*.

Zhang, C.; Platt, J. C.; and Viola, P. A. 2005. Multiple instance boosting for object detection. In *NIPS*, 1417–1424.

Zou, J. Y.; Chaudhuri, K.; and Kalai, A. T. 2015. Crowdsourcing feature discovery via adaptively chosen comparisons. *HCOMP*.